



# C++ KURSPROJEK

*Karlskrona, 2025*

# Presentation av mig och min projekt

---

- **Namn:** Ha Thanh Chu
- **Projekt:** Programvara  
försäljningshantering för  
klädbutiker
- **Link Github:** <https://github.com/Me-Bim-Bi/Project-C->



# Innehållet

1. Kortfattad beskrivning av projektet och demokörning
2. Beskrivning av uppbyggnaden med utgångspunkt från klassdiagrammet
3. I koden visa att min lösning uppfyller
  - de specifika kraven
  - tilläggskrav för högre betyg
4. Att besvara frågor och redogöra för delar av min lösning



# 1. Kortfattad beskrivning av projektet och demokörning

Ett program utvecklat för att hantera produkter, anställda och beräkna bokslut i en klädaffär

```
*****
<<<<<<<<<<< Welcome To The Fashion Store! >>>>>>>>>>>
*****
|                                     MENU                                     |
|-----|
|1. Import a clothing                ||11. Find an employee          |
|2. Import a cosmetic                ||12. Remove an employee        |
|3. Find a product                  ||13. Edit an employee's ID     |
|4. Remove a product                ||14. Edit an employee's information|
|5. Edit a product's id             ||15. Show all employees        |
|6. Edit a product's information    ||16. Save employees' list to file |
|7. Sell a product                 ||17. Show financial statement   |
|8. Show all products              ||18. Save the statement in file  |
|9. Save products' list to file    ||19. Exit                      |
|10. Add an employee               |
|-----|
|                                     *** Please select from the menu above: *** |
|-----|
```

# 1. Kortfattad beskrivning av projektet och demokörning

## □ Produktregler:

- Varje produkt har ett unikt ID.
- Vid tillägg av en produkt:
  - ✓ Om produkten redan finns i systemet kan användaren välja att uppdatera kvantiteten genom att importera mer eller låta den vara oförändrad.
  - ✓ En ny produkt kan endast läggas till om dess ID inte redan finns i systemet.

## □ Anställdsregler:

- Varje anställd har ett unikt ID.
- En anställd kan endast läggas till om deras ID inte redan finns i systemet.

```
*****  
<<<<<<<<<< Welcome To The Fashion Store! >>>>>>>>>>>  
*****  
| | MENU | |  
-----| |  
| 1. Import a clothing | | 11. Find an employee | |  
| 2. Import a cosmetic | | 12. Remove an employee | |  
| 3. Find a product | | 13. Edit an employee's ID | |  
| 4. Remove a product | | 14. Edit an employee's information | |  
| 5. Edit a product's id | | 15. Show all employees | |  
| 6. Edit a product's information | | 16. Save employees' list to file | |  
| 7. Sell a product | | 17. Show financial statement | |  
| 8. Show all products | | 18. Save the statement in file | |  
| 9. Save products' list to file | | 19. Exit | |  
| 10. Add an employee | |-----| |  
-----| |  
| | *** Please select from the menu above: *** | |  
-----| |
```

# 1. Kortfattad beskrivning av projektet och demokörning

## □ Sälja varor:

- Försäljningen genomförs endast när följande villkor är uppfyllda:
  - ✓ **Rätt artikel-ID** som finns i lager.
  - ✓ **Giltigt medarbetar-ID**: korrekt och finns registrerat i systemet.
  - ✓ **Tillräckligt lager**: Det måste finnas ett tillräckligt antal artiklar i lager för att genomföra försäljningen.
- När en vara säljs minskas varans lagerkvantitet, och en försäljningsprovision tilldelas säljaren som genomförde försäljningen.

```
*****  
<<<<<<<<<<<< Welcome To The Fashion Store! >>>>>>>>>>>>  
*****  
| | MENU | |  
-----| |  
| 1. Import a clothing | | 11. Find an employee | |  
| 2. Import a cosmetic | | 12. Remove an employee | |  
| 3. Find a product | | 13. Edit an employee's ID | |  
| 4. Remove a product | | 14. Edit an employee's information | |  
| 5. Edit a product's id | | 15. Show all employees | |  
| 6. Edit a product's information | | 16. Save employees' list to file | |  
| 7. Sell a product | | 17. Show financial statement | |  
| 8. Show all products | | 18. Save the statement in file | |  
| 9. Save products' list to file | | 19. Exit | |  
| 10. Add an employee | |-----| |  
| | *** Please select from the menu above: *** | |  
-----| |
```

## **1. Kortfattad beskrivning av projektet och demokörning**

Hitta, redigera eller ta bort med ID:

- ID-nummer kan endast ändras om det nya ID-numret inte redan existerar i systemet.

## Bekräftelse före ändringar:

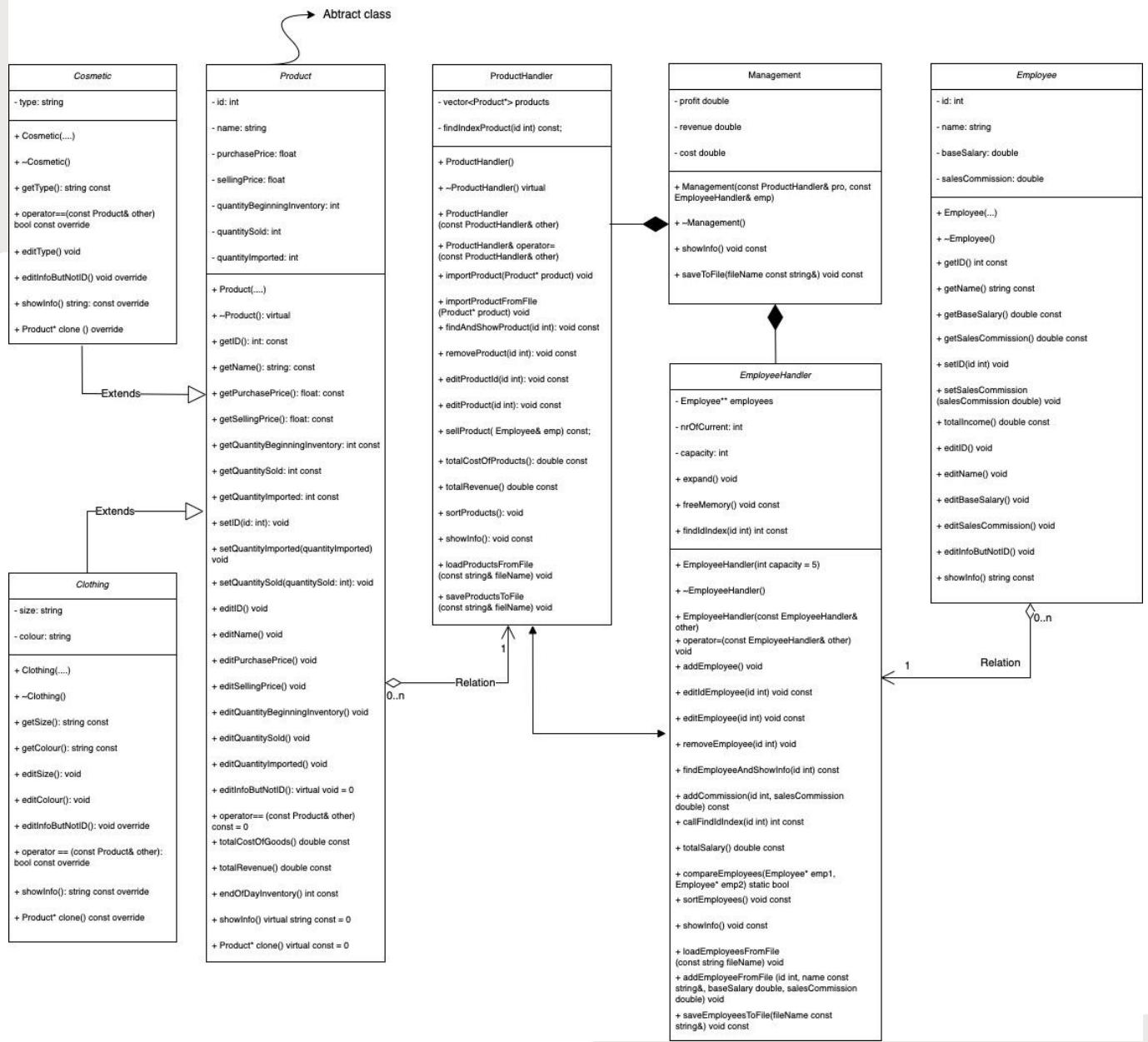
- När information om en produkt eller anställd ska redigeras eller tas bort, visas dess detaljer först för att ge användaren möjlighet att bekräfta eller avbryta ändringen.

## Datasortering:

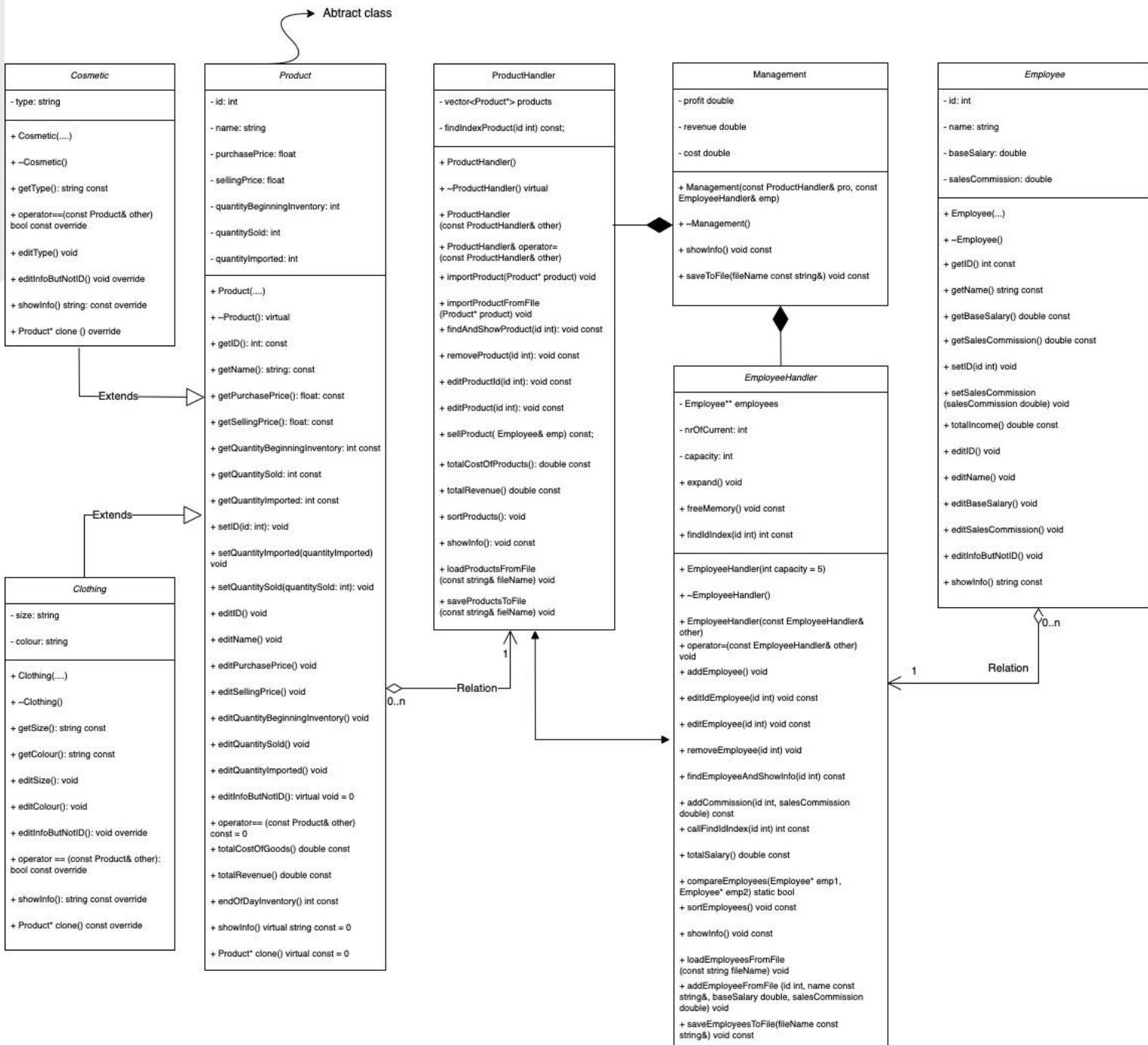
- Data sorteras innan den visas på skärmen eller sparas i en fil.
    - ✓ Med product: Clothing -> Cosmetic -> ID
    - ✓ Med anställd: name -> ID

# **1. Kortfattad beskrivning av projektet och demokörning**

## 2. Beskrivning av uppbyggnaden med utgångspunkt från klassdiagrammet



```
*****
***** Welcome To The Fashion Store! *****
*****
|           MENU           |
|-----|
|1. Import a clothing      ||11. Find an employee|
|2. Import a cosmetic       ||12. Remove an employee|
|3. Find a product         ||13. Edit an employee's ID|
|4. Remove a product        ||14. Edit an employee's information|
|5. Edit a product's id     ||15. Show all employees|
|6. Edit a product's information||16. Save employees' list to file|
|7. Sell a product          ||17. Show financial statement|
|8. Show all products        ||18. Save the statement in file|
|9. Save products' list to file||19. Exit|
|10. Add an employee         |
|-----|
| *** Please select from the menu above: *** |
```



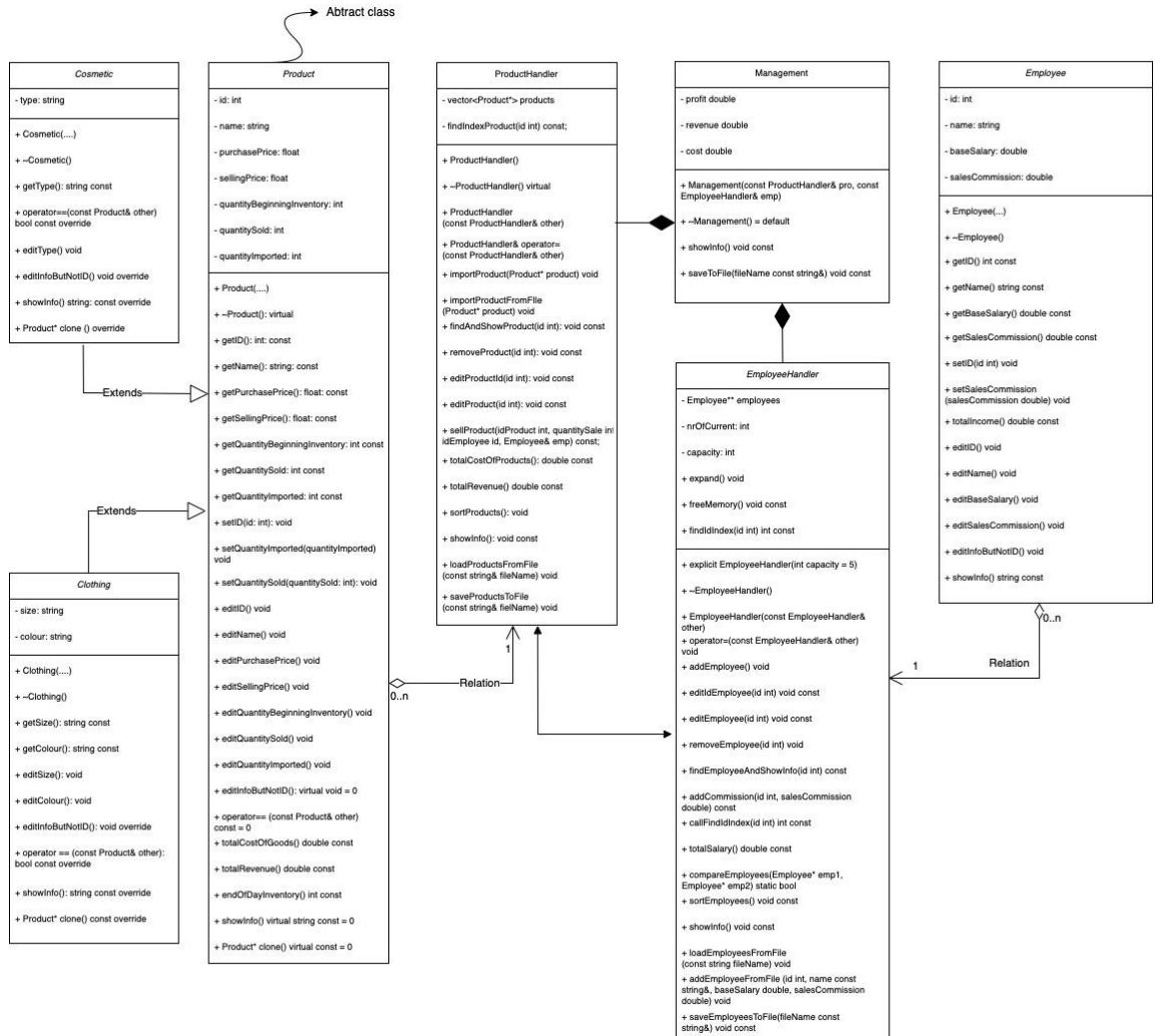
### **3. I koden visa att min lösning uppfyller kraven**

3a. De specifika kraven

3b. Tilläggskrav för högre betyg

# 3a. De specifika kraven

✓ Ett visst antal, minst 5, egendefinierade relevanta klasser



# 3a. De specifika kraven

- ✓ Ett visst antal, minst 5, egendefinierade relevanta klasser
- ✓ **Flera objekt av någon av de egendefinierade klasserna ska finnas. Dessa ska hanteras i en behållare (array, vector, ...).**

```
class ProductHandler {  
private:  
    vector<Product*> products;  
    int findIndexProduct(int id) const;  
public:  
  
class EmployeeHandler {  
private:  
    //The number of employees does not fluctuate too much  
    //so it is more suitable to use a dynamic pointer.  
    Employee* *employees;  
    int nrOfCurrent;  
    int capacity;  
    void expand();  
    void freeMemory();  
    int findIdIndex(int id) const;
```

# 3a. De specifika kraven

- ✓ Ett visst antal, minst 5, egendefinierade relevanta klasser
- ✓ Flera objekt av någon av de egendefinierade klasserna ska finnas. Dessa ska hanteras i en behållare (array, vector, ...).
- ✓ **Egendefinierat logiskt och rimligt använd med abstrakt klass(er)**

```
class Cosmetic : public Product {  
private:  
    string type;  
public:  
    Cosmetic(int id = -1, const string& name = "?", float purchasePrice = 0, float sellingPrice = 0, int quantityBeginningInventory = 0, int quantitySold = 0, int quantityImported = 0, const string& info = "");  
    ~Cosmetic() override;  
  
    string getType() const;  
  
    void editType();  
    void editInfoButNotID() override;  
  
    bool operator==(const Product &other) const override;  
  
    string showInfo() const override;  
    Product* clone() const override;
```

```
class Product {  
private:  
    int id;  
    string name;  
    float purchasePrice;  
    float sellingPrice;  
    int quantityBeginningInventory;  
    int quantitySold;  
    int quantityImported;  
public:  
    Product(int id = -1, const string& name = "?", float purchasePrice = 0, float sellingPrice = 0, int quantityBeginningInventory = 0, int quantitySold = 0, int quantityImported = 0, const string& info = "");  
    ~Product() override;  
  
    virtual bool operator==(const Product& other) const;  
  
    void editId();  
    void editName();  
    void editPurchasePrice();  
    void editSellingPrice();  
    void editQuantityBeginningInventory();  
    void editQuantitySold();  
    void editQuantityImported();  
    virtual void editInfoButNotID() = 0;  
  
    double totalCostOfGoods() const;  
    double totalRevenue() const;  
    int endOfDayInventory() const;  
  
    virtual string showInfo() const = 0;  
    virtual Product* clone() const = 0;
```

```
class Clothing : public Product{  
private:  
    string size;  
    string colour;  
public:  
    Clothing(int id = -1, const string& name = "?", float purchasePrice = 0, float sellingPrice = 0, int quantityBeginningInventory = 0, int quantitySold = 0, int quantityImported = 0, const string& info = "");  
    ~Clothing() override;  
  
    string getSize() const;  
    string getColour() const;  
  
    void editSize();  
    void editColour();  
    void editInfoButNotID() override;  
  
    bool operator==(const Product &other) const override;  
  
    string showInfo() const override;  
    Product* clone() const override;
```

# 3a. De specifika kraven

---

- ✓ Ett visst antal, minst 5, egendefinierade relevanta klasser
- ✓ Flera objekt av någon av de egendefinierade klasserna ska finnas. Dessa ska hanteras i en behållare (array, vector, ...).
- ✓ Egendefinierat logiskt och rimligt arv med abstrakt klass/er
- ✓ **Överskuggning och dynamisk bindning** på ett lämpligt och relevant sätt

```
void ProductHandler::showInfo() const {
    int index = 1;
    cout << "\nProduct in store now:\n";
    for (auto product : products) {
        cout << index << ". " << product->showInfo() << endl;
        index++;
    }
}
```

```
string showInfo() const override;
Product* clone() const override;

string Cosmetic::showInfo() const {
    return "Cosmetic: " + Product::showInfo() + ", type: " + type;
}

Product * Cosmetic::clone() const {
    return new Cosmetic(*this);
}
```

```
virtual string showInfo() const = 0;
virtual Product* clone() const = 0;

string Product::showInfo() const {
    ostringstream stringInfo;
    stringInfo << fixed << setprecision(2);
    stringInfo << "ID: " << id
    << ", name: " << name
    << ", purchase price: " << purchasePrice
    << ", selling price: " << sellingPrice
    << ", beginning inventory: " << quantityBeginningInventory
    << ", number of goods imported: " << quantityImported
    << ", number of goods sold: " << quantitySold
    << ", number of goods in stock: " << endOfDayInventory();
    return stringInfo.str();
}
```

```

}
else {
    product->editInfoButNotID();
    products.push_back(x: product->clone());
    cout << "The product with id: " << idProduct << " has been imported into the wa
}
```

```
string showInfo() const override;
Product* clone() const override;

string Clothing::showInfo() const {
    return "Clothing: " + Product::showInfo() + ", size: " + size + ", colour " + colour;
}
```

# 3a. De specifika kraven

- ✓ Rimlig användning av dynamisk minneshantering (genom användande av new och delete)

```
employees[this->nrOfCurrent] = new Employee(id, name, baseSalary, salesCommission);
employees[this->nrOfCurrent]->editInfoButNotID();
nrOfCurrent++;
}
```

```
void EmployeeHandler::expand() {
    this->capacity += 5;
    auto** temp = new Employee* [this->capacity]{nullptr};
    for(int i = 0; i < this->nrOfCurrent; i++) {
        temp[i] = this->employees[i];
    }
    delete[] this->employees;
    this->employees = temp;
}

void EmployeeHandler::freeMemory() {
    for (int i = 0; i < nrOfCurrent; i++) {
        delete employees[i];
    }
    delete[] employees;
    employees = nullptr;
}
```

# 3a. De specifika kraven

- ✓ Rimlig användning av dynamisk minnes hantering (genom användande av new och delete)
- ✓ Någon datastruktur från standardbiblioteket (**vector, stack, kö, ...**)

```
else {
    product->editInfoButNotID();
    products.push_back(*product->clone());
    cout << "The product with id: " << idProduct << " has been imported into the warehouse." << endl;
}
```

```
if (askYesNo(question: "Do you want to remove this product? ")) {
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    products.erase(position: products.begin() + index);
    cout << "The product has been deleted." << endl;
}
```

```
ProductHandler::~ProductHandler() {
    for (auto product : products) {
        delete product;
    }
}
```

```
ProductHandler::ProductHandler(const ProductHandler &other) {
    for (auto product : other.products) {
        products.push_back(*product->clone());
    }
}
```

# 3a. De specifika kraven

- ✓ Rimlig användning av dynamisk minnes hantering (genom användande av new och delete)
- ✓ Någon datastruktur från standardbibliot eket (vector, stack, kö, ...)
- ✓ **Läsning från och/eller skrivning till textfiler**

```
int main() {
    //_CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);
    EmployeeHandler employeeHandlerPointers;
    ProductHandler productHandlerVector;
    employeeHandlerPointers.loadEmployeesFromFile(fileName: "All Employees");
    productHandlerVector.loadProductsFromFile(fileName: "All Products");
```

```
void sortProducts();
void showInfo () const;

void loadProductsFromFile(const string& fileName);
void saveProductsToFile(const string& fileName) const;
```

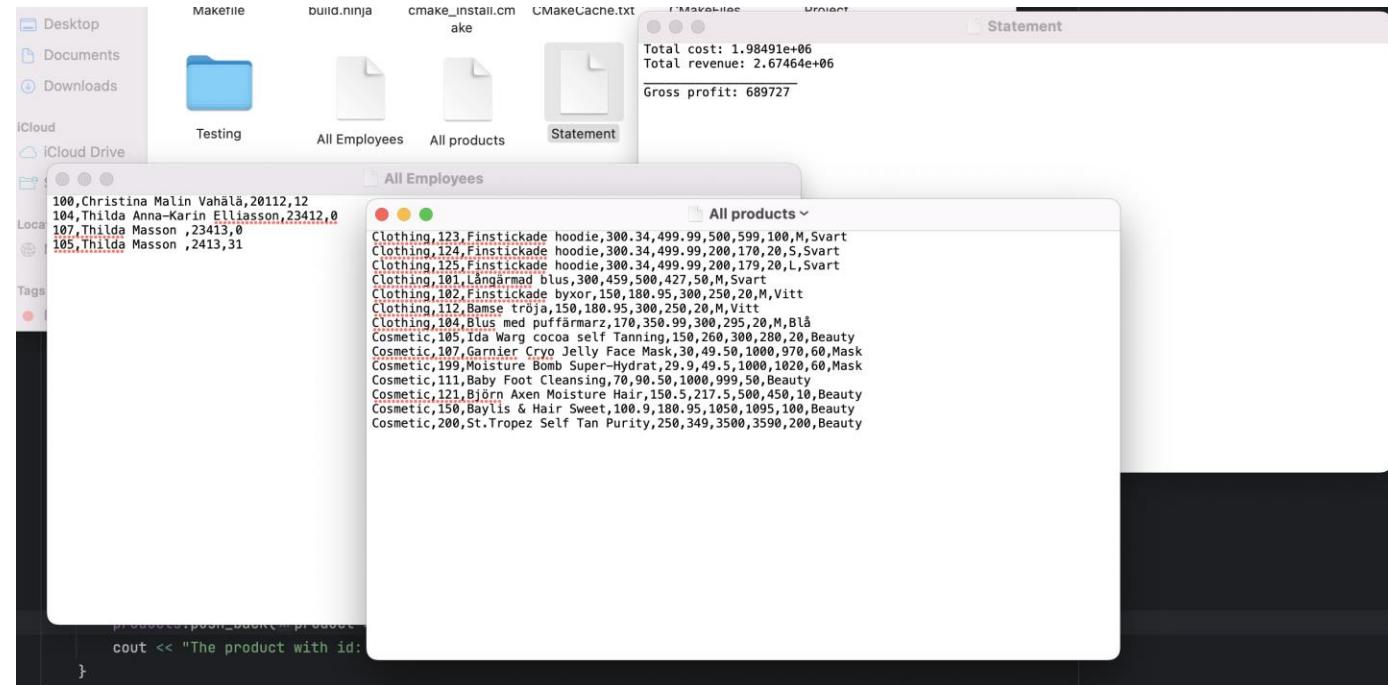
```
void sortEmployees() const;
void showInfo() const;

void loadEmployeesFromFile(const string& fileName);
void addEmployeeFromFile(int id, const string& name, double baseSalary, double salesCommission);
void saveEmployeesToFile(const string& fileName) const;
```

```
void saveToFile(const string& fileName) const;
```

# 3a. De specifika kraven

- ✓ Rimlig användning av dynamisk minnes hantering (genom användande av new och delete)
- ✓ Någon datastruktur från standardbibliot eket (vector, stack, kö, ...)
- ✓ **Läsning från och/eller skrivning till txtfiler**



The screenshot shows a macOS desktop environment. In the foreground, a terminal window displays a C++ program's output. The code reads from a file named 'products.txt' and prints products with ID 100 and 101. The output is:

```
products.txt
cout << "The product with id: "
}
The product with id: 100
The product with id: 101
```

In the background, there is a file browser window titled 'All Employees' and another window titled 'All products'. Both windows show lists of items with various details like name, price, and category. There are also several files on the desktop, including 'Makefile', 'build.ninja', 'cmake\_install.cmake', 'CMakeCache.txt', 'Statement', and 'Testing'. A status bar at the bottom of the screen shows the total cost, revenue, and profit.

# 3a. De specifika kraven

---

- ✓ Koden ska överensstämma med klassdiagrammet, dvs de relationer som finns i klassdiagrammet ska vara implementerade
- ✓ I normalfallet ska alla medlemsvariabler vara privata.



```

class Employee {
private:
    int id;
    string name;
    double baseSalary;
    double salesCommission;

public:
    Employee(int id = 0, string name = "?", double ba
    ~Employee();

    int getId() const;
    string getName() const;
    double getBaseSalary() const;
    double getSalesCommission() const;

    void setId(int id);
    void setSalesCommission(double salesCommission);

    double totalIncome() const;

    void editID();
    void editName();
    void editBaseSalary();
    void editSalesCommission();
    void editInfoButNotID();
    string showInfo() const;
};


```

```

class EmployeeHandler {
private:
    //The number of employees does not fluctuate too much
    //so it is more suitable to use a dynamic pointer.
    Employee* *employees;
    int nrOfCurrent;
    int capacity;
    void expand();
    void freeMemory();
    int findIdIndex(int id) const;
public:
    EmployeeHandler(int capacity = 5);
    ~EmployeeHandler();
    EmployeeHandler(const EmployeeHandler& other);
    EmployeeHandler& operator=(const EmployeeHandler& other);

    void addEmployee();
    void editIdEmployee(int id) const;
    void editEmployee(int id) const;
    void removeEmployee(int id);
    void findEmployeeAndShowInfo(int id) const;
    void addCommission(int id, double salesCommission) const;

    int callFindIdIndex(int id) const;

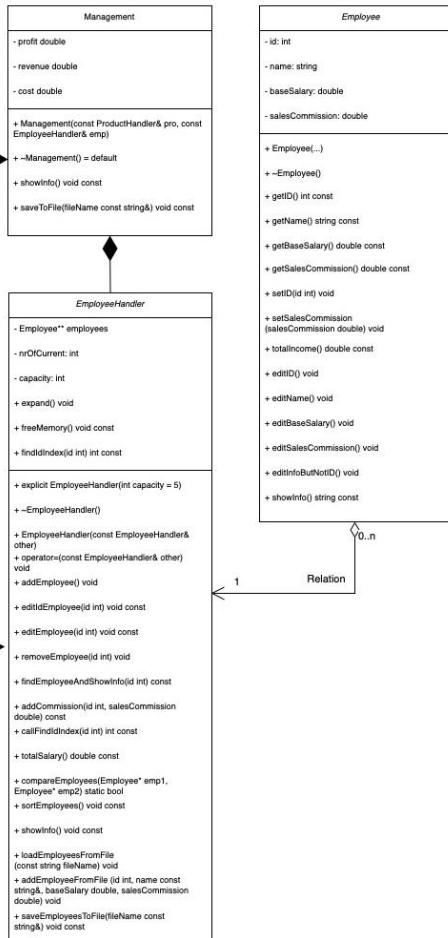
    double totalSalary() const;

    static bool compareEmployees(Employee* emp1, Employee* emp2);
    void sortEmployees() const;
    void showInfo() const;

    void loadEmployeesFromFile(const string& fileName);
    void addEmployeeFromFile(int id, const string& name, double baseSalary,
    void saveEmployeesToFile(const string& fileName) const;

};


```



```

class Management {
private:
    double profit;
    double revenue;
    double cost;
public:
    Management(const ProductHandler& pro, const EmployeeHandler& emp);
    ~Management();

    void showInfo() const;
    void saveToFile(const string& fileName) const;
};


```

```

class ProductHandler {
private:
    vector<Product*> products;
    int findIndexProduct(int id) const;
public:
    ProductHandler();
    virtual~ProductHandler();
    ProductHandler(const ProductHandler& other);
    ProductHandler& operator=(const ProductHandler& other);

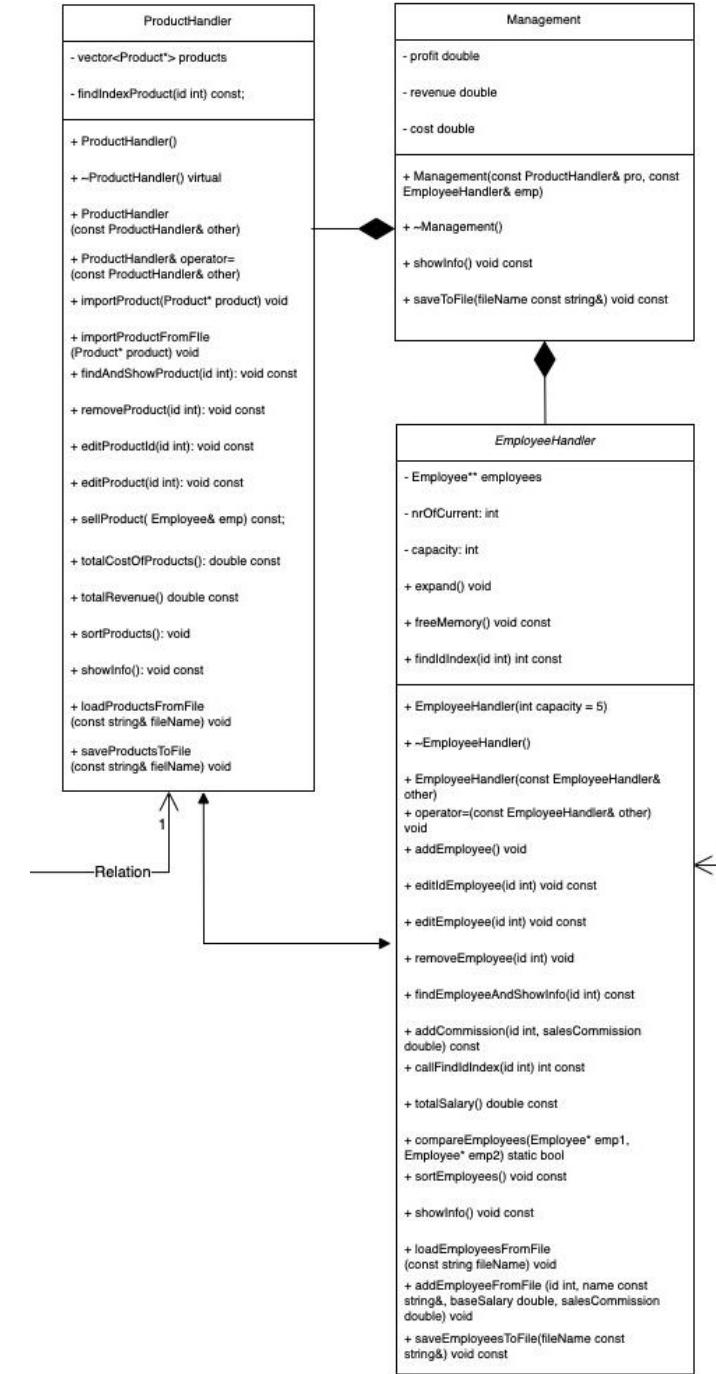
    void importProduct(Product* product);
    void importProductFromFile(const Product* product);
    void findAndShowProduct(int id) const;
    void removeProduct(int id);
    void editProductId(int id) const;
    void editProduct(int id) const;

    void sellProduct(const EmployeeHandler& emp) const;
    double totalCostOfProducts() const;
    double totalRevenue() const;

    void sortProducts();
    void showInfo () const;

    void loadProductsFromFile(const string& fileName);
    void saveProductsToFile(const string& fileName) const;
};


```



```

class EmployeeHandler {
private:
    //The number of employees does not fluctuate too much
    //so it is more suitable to use a dynamic pointer.
    Employee* *employees;
    int nrOfCurrent;
    int capacity;
    void expand();
    void freeMemory();
    int findIdIndex(int id) const;
public:
    EmployeeHandler(int capacity = 5);
    ~EmployeeHandler();
    EmployeeHandler(const EmployeeHandler& other);
    EmployeeHandler& operator=(const EmployeeHandler& other);

    void addEmployee();
    void editIdEmployee(int id) const;
    void editEmployee(int id) const;
    void removeEmployee(int id);
    void findEmployeeAndShowInfo(int id) const;
    void addCommission(int id, double salesCommission) const;

    int callFindIdIndex(int id) const;

    double totalSalary() const;

    static bool compareEmployees(Employee* emp1, Employee* emp2);
    void sortEmployees() const;
    void showInfo() const;

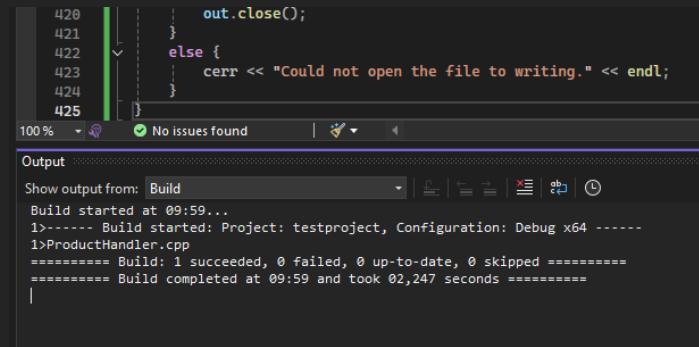
    void loadEmployeesFromFile(const string& fileName);
    void addEmployeeFromFile(int id, const string& name, double baseSalary, double salesCommission) const;
    void saveEmployeesToFile(const string& fileName) const;
}

```

# 3a. De specifika kraven

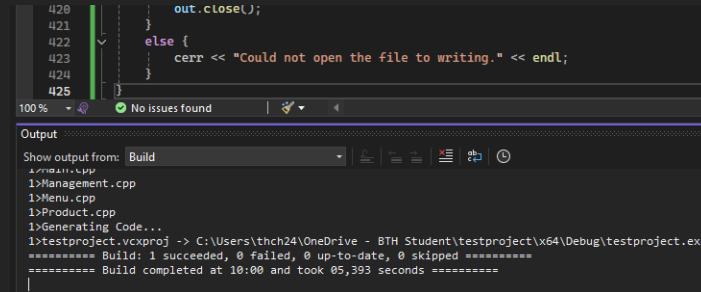
---

- ✓ Koden ska överensstämma med klassdiagrammet, dvs de relationer som finns i klassdiagrammet ska vara implementerade
- ✓ I normalfallet ska alla medlemsvariabler vara privata.
- ✓ **Koden ska vara "städad", dvs det ska inte finnas utkommenterade kodavsnitt.**



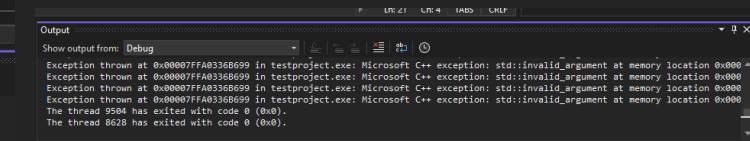
```
420     |     out.close();
421     | }
422     | else {
423     |     cerr << "Could not open the file to writing." << endl;
424 }
425 }
```

100% No issues found



```
420     |     out.close();
421     | }
422     | else {
423     |     cerr << "Could not open the file to writing." << endl;
424 }
425 }
```

100% No issues found



Output Show output from: Debug

```
Exception thrown at 0x00007FFA0336B699 in testproject.exe: Microsoft C++ exception: std::invalid_argument at memory location 0x0000000000000000.
Exception thrown at 0x00007FFA0336B599 in testproject.exe: Microsoft C++ exception: std::invalid_argument at memory location 0x0000000000000000.
Exception thrown at 0x00007FFA0336B699 in testproject.exe: Microsoft C++ exception: std::invalid_argument at memory location 0x0000000000000000.
The thread 9504 has exited with code 0 (0x0).
The thread 8628 has exited with code 0 (0x0).
```

Inga minnesläckor

# 3a. De specifika kraven

---

- ✓ Koden ska överensstämma med klassdiagrammet, dvs de relationer som finns i klassdiagrammet ska vara implementerade
- ✓ I normalfallet ska alla medlemsvariabler vara privata.
- ✓ **Koden ska vara "städad", dvs det ska inte finnas utkommenterade kodavsnitt.**
- ✓ **Try to DRY (min)**

```

//The number of employees does not fl
//so it is more suitable to use a dyn
Employee* *employees;
int nrOfCurrent;
int capacity;
void expand();
void freeMemory();
int findIdIndex(int id) const;
public:
EmployeeHandler(int capacity = 5);
~EmployeeHandler();
EmployeeHandler(const EmployeeHandler& other);
EmployeeHandler& operator=(const EmployeeHandler& other);

int getNrOfCurrent() const;

void addEmployee();
void editIdEmployee(int id) const;
void editEmployee(int id) const;
void removeEmployee(int id);
void findEmployeeAndShowInfo(int id)
void addCommission(int id, double sal

int callFindIdIndex(int id) const;

```

Project ~/Workspaces/Project-C/F

- cmake-build-debug
  - Clothing.cpp
  - Clothing.h
  - CMakeLists.txt
  - Cosmetic.cpp
  - Cosmetic.h
  - Employee.cpp
  - Employee.h
  - EmployeeHandler.cpp
  - EmployeeHandler.h
  - FuntionToEditInformation.h
  - main.cpp
  - Management.cpp
  - Management.h
  - Menu.cpp
  - Menu.h
  - Product.cpp
  - Product.h
  - ProductHandler.cpp
  - ProductHandler.h
- External Libraries
- Scratches and Consoles

Employee.h EmployeeHandler.h

```

9  using namespace std;
10 //this function template
11 //ensuring that the data
12 template <typename T>
13 void editPrice(const string&
14   while (true) {
15     string input;
16     cout << textToInpu
17     getline([&] cin, [&]
18
19     try {
20       size_t onlyNum;
21       T temp;
22       if constexpr (is
23         //Assign
24         temp = in
25     } else if (con
26       //convert
27       long long onlyI
28       if (onlyI
29         throw
30       if (temp >
31         throw
32       if (temp >
33         throw
34       if (temp >
35         throw
36       if (temp >
37         throw
38       if (temp >
39         temp = st
40       if (temp >
41         cout << "C
42         cont
43     }
44   }
45   else if (con

```

```

class ProductHandler {
private:
vector<Product*> products;
int findIndexProduct(int i
public:
#ifndef MENU_H
#define MENU_H
#include ...

using namespace std;

void Menu();
int checkInputDataInt();
bool askYesNo(const string &question);
int checkMenuChoice();
void getMenuChoice(int answer, ProductHandler& productHandlerVector);
int validateAndConvertToInt(const string& str, const string& errorMessage);
float validateAndConvertToFloat(const string& str, const string& errorMessage);
double validateAndConvert.ToDouble(const string& str, const string& errorMessage);

#endif //MENU_H

```

# 3a. De specifika kraven

---

- ✓ Koden ska överensstämma med klassdiagrammet, dvs de relationer som finns i klassdiagrammet ska vara implementerade
- ✓ I normalfallet ska alla medlemsvariabler vara privata.
- ✓ Koden ska vara "städad", dvs det ska inte finnas utkommenterade kodavsnitt.
- ✓ Try to DRY (min)
- ✓ **Numeriska värden (talkonstanter)  
ska i möjligaste mån representeras med egendefinierade konstanter eller enum.  
Detta gäller naturligtvis inte defaultvärden**

```

EmployeeHandler(const EmployeeHandler& other),
EmployeeHandler& operator=(const EmployeeHandler& other);

int getNrOfCurrent() const;

void addEmployee();
void editIdEmployee(int id) const;
void editEmployee(int id) const;
void removeEmployee(int id);
void findEmployeeAndShowInfo(int id) const;
void addCommission(int id, double salesCommission) const;

int callFindIdIndex(int id) const;

double totalSalary() const;

// Hàm so sánh nhân viên: so sánh tất cả các từ trong tên, nếu tên giống nhau thì so sánh id giảm dần
static bool compareEmployees(Employee* emp1, Employee* emp2);
void sortEmployees() const;
void showInfo() const;

void loadEmployeesFromFile(const string& fileName);

```

```

private:
    double profit;
    double revenue;
    double cost;

public:
    Management(const ProductHandler& pro, const EmployeeHandler& emp);
    ~Management();

    void showInfo() const;
    void saveToFile(const string& fileName) const;
};

```

```

    string getName() const;
    float getPurchasePrice() const;
    float getSellingPrice() const;
    int getQuantityBeginningInventory() const;
    int getQuantitySold() const;
    int getQuantityImported() const;

    void setId(int id);
    void setQuantitySold(int quantitySold);
    void setQuantityImported(int quantityImported);

    virtual bool operator==(const Product& other) const;

    void editId();
    void editName();
    void editPurchasePrice();
    void editSellingPrice();
    void editQuantityBeginningInventory();
    void editQuantitySold();
    void editQuantityImported();
    virtual void editInfoButNotID() = 0;

    double totalCostOfGoods() const;
    double totalRevenue() const;
    int endOfDayInventory() const;

```

```

class Clothing : public Product
private:
    string size;
    string colour;
public:
    Clothing(int id = -1, const string& name = "?", float p = 0, int quantitySold = 0, int quantityImported = 0, const string& desc = "");
    void editSize();
    void editColour();
    void editInfoButNotID() override;

    bool operator==(const Product &other) const override;

    string showInfo() const override;
};

Product* clone() const override;

```

```

class Cosmetic : public Product {
private:
    string type;
public:
    Cosmetic(int id = -1, const string& name = "?", int quantitySold = 0, int quantityImported = 0, const string& desc = "");
    ~Cosmetic() override;

    string getType() const;

    void editType();
    void editInfoButNotID() override;

    bool operator==(const Product &other) const override;

    string showInfo() const override;
    Product* clone() const override;
};

private:
    vector<Product*> products;
    int findIndexProduct(int id) const;
public:
    ProductHandler();
    virtual ~ProductHandler();
    ProductHandler(const ProductHandler& other);
    ProductHandler& operator=(const ProductHandler& other);

    void importProduct(Product* product);
    void importProductFromFile(const Product* product);
    void findAndShowProduct(int id) const;
    void removeProduct(int id);
    void editProductId(int id) const;
    void editProduct(int id) const;

    void sellProduct(const EmployeeHandler& emp) const;
    double totalCostOfProducts() const;
    double totalRevenue() const;

    void sortProducts();
    void showInfo() const;
};

```

# 3a. De specifika kraven

---

- ✓ Koden ska överensstämma med klassdiagrammet, dvs de relationer som finns i klassdiagrammet ska vara implementerade
- ✓ I normalfallet ska alla medlemsvariabler vara privata.
- ✓ Koden ska vara "städad", dvs det ska inte finnas utkommenterade kodavsnitt.
- ✓ Try to DRY (min)
- ✓ Numeriska värden (talkonstanter)  
ska i möjligaste mån representeras med egendefinierade konstanter eller enum.  
Detta gäller naturligtvis inte defaultvärden
- ✓ **Allt allokerat minne ska vara frigjort vid programslut.**

```
ProductHandler::~ProductHandler() {
    for (auto product : products) {
        delete product;
    }
}
```

```
~Clothing() override;
```

```
~Cosmetic() override;
```

```
virtual ~Product();
```

```
void EmployeeHandler::freeMemory() {
    for (int i = 0; i < nrOfCurrent; i++) {
        delete employees[i];
    }
    delete[] employees;
    employees = nullptr;
}
```

```
~Employee();
```

```
~Management();
```

# 3a. De specifika kraven

---

- ✓ Koden ska överensstämma med klassdiagrammet, dvs de relationer som finns i klassdiagrammet ska vara implementerade
- ✓ I normalfallet ska alla medlemsvariabler vara privata.
- ✓ Koden ska vara "städad", dvs det ska inte finnas utkommenterade kodavsnitt.
- ✓ Try to DRY (min)
- ✓ Numeriska värden (talkonstanter)  
ska i möjligaste mån representeras med egendefinierade konstanter eller enum.  
Detta gäller naturligtvis inte defaultvärden
- ✓ Allt allokerat minne ska vara frigjort vid programslut.
- ✓ **Programmet ska inte krascha eller låsas.**

```

if(productInformation.size() < 9) { //show the error message if the input data has not enough
    cerr << "Error on line: " << lineNumber << ". Unknown product format. Skipping..." << endl;
    continue;
}

string productType= productInformation[0], name = productInformation[2];
try {
    int id = validateAndConvertToInt(str:productInformation[1],
        errorMessage: "ID must be an integer");
    float purchasePrice = validateAndConvertToFloat(str:productInformation[3],
        errorMessage: "Purchase price must be a float");
    float sellingPrice = validateAndConvertToFloat(str:productInformation[4],
        errorMessage: "Selling price must be a float");
    int quantityBeginningInventory = validateAndConvertToInt(str:productInformation[5],
        errorMessage: "Beginning inventory quantity must be an integer");
    int quantitySold = validateAndConvertToInt(str:productInformation[6],
        errorMessage: "Sold quantity must be an integer");
    int quantityImported = validateAndConvertToInt(str:productInformation[7],
        errorMessage: "Imported quantity must be an integer");

unique_ptr<Product> product; //create a smart pointer to make everything easier

if(productType == "Clothing") { //check if product type is clothing
    if(productInformation.size() != 10) {
        cerr << "Error on line: " << lineNumber << ". Invalid product type." << endl;
        continue;
    }
    const string& size = productInformation[8];
    const string& colour = productInformation[9];
    product = make_unique <Clothing> (id, name, purchasePrice, sellingPrice,
        quantitySold, quantityImported, size, colour);
}
else if(productType == "Cosmetic") {
    if(productInformation.size() != 10) {
        cerr << "Error on line: " << lineNumber << ". Invalid product type." << endl;
        continue;
    }
    const string& brand = productInformation[8];
    const string& type = productInformation[9];
    product = make_unique <Cosmetic> (id, name, purchasePrice, sellingPrice,
        quantitySold, quantityImported, brand, type);
}
else {
    cerr << "Error on line: " << lineNumber << ". Invalid product type." << endl;
    continue;
}
}

```

C	D	E	F	G	H
Expected results	Actual result	Pass	Fail	Reason	Fix
gick inte	Error: The input data is not valid!Please try again!	x			
gick inte	Error: The input data is not valid!Please try again!	x			
gick inte	Error: The input data is not valid!Please try again!	x		cin.fail() is not fully reset & cin.ignore() does not completely remove redundant data	Reset cin's error state using cin.clear(). Remove all redundant data from the stream using cin.ignore(numeric_limits<streamsize>::max(), '\n').
12.34	kom till menyval 12	x		ha inte kollat om det är float nummer	if(input.find_first_not_of("0123456789") != string::npos){ throw invalid_argument("The input data is not valid!"); }
19 1	kom till menu 19, exit kom till menu 1, insert a product	x	x		
	yes/no question				
1 2 .3 YES 13 4 NO 14 5 Yes	ok ok ok ok ok	ok ok ok ok ok	x x x x x		

# 3b. Tilläggsskrav för högre betyg

- ✓ Relevant användande av Lamdautryck

```
void ProductHandler::sortProducts() {
    try {
        // Check for duplicate IDs before sorting
        for (size_t i = 0; i < products.size(); ++i) {
            for (size_t j = i + 1; j < products.size(); ++j) {
                if (products[i]->getID() == products[j]->getID()) {
                    throw runtime_error("Duplicate product ID found: " + to_string(val:products[i]->getID()));
                }
            }
        }
        // Sorting the products by type (Clothing before Cosmetic) and by ID in descending order
        sort(first: products.begin(), last: products.end(), comp: [](Product* p1, Product* p2)->bool {
            // Check if the product type is valid (either Clothing or Cosmetic)
            // this product format and id duplication check is mainly to anticipate and handle
            //possible errors.
            if (typeid(*p1) != typeid(Clothing) && typeid(*p1) != typeid(Cosmetic)) {
                throw runtime_error("Invalid product type: " + string(s:typeid(*p1).name()));
                // using string to convert typeid(*(p1).name from const. char* to string
            }
            if (typeid(*p2) != typeid(Clothing) && typeid(*p2) != typeid(Cosmetic)) {
                throw runtime_error("Invalid product type: " + string(s:typeid(*p2).name()));
            }

            // Arrange clothing before cosmetic
            if (typeid(*p1) == typeid(Clothing) && typeid(*p2) != typeid(Clothing)) {
                return true;
            }
            if (typeid(*p2) == typeid(Clothing) && typeid(*p1) != typeid(Clothing)) {
                return false;
            }
            return p1->getID() > p2->getID(); // Sort by ID in descending order
        });
    } catch (const exception& e) {
        cerr << "Error during sorting: " << e.what() << endl;
    }
}
```

## 3b. Tilläggsskrav för högre betyg

- ✓ Relevant användande av Lamdautryck
- ✓ **Användande av undantagshantering där detta är lämpligt**

```
try {  
    int id = validateAndConvertToInt(str:employeeInformation[0],  
        errorMessage: "ID must be an integer");  
    const string& name = employeeInformation[1];  
    double baseSalary = validateAndConvert.ToDouble(str:employeeInformation[2],  
        errorMessage: "Base salary must be a number");  
    double salesCommission = validateAndConvert.ToDouble(str:employeeInformation[3],  
        errorMessage: "Sales commission must be a number");  
  
    //add employee if the data is correct type  
    addEmployeeFromFile(id, name, baseSalary, salesCommission);  
}  
  
//show the error  
catch (exception& e){  
    cerr << "Error in line: " << lineNumber << " in file " << fileName << ". " << e.what() << " .Skipping..." << endl;
```

# 3b. Tilläggskrav för högre betyg

- ✓ Relevant användande av Lamdautryck
- ✓ Användande av undantagsshantering där detta är lämpligt
- ✓ **Användande av smarta pekare där detta är lämpligt**

```
unique_ptr<Product> product; //create a smart pointer to make everything easier

if(productType == "Clothing") {    //check if product type is Clothing to add product
    if(productInformation.size() != 10) {
        cerr << "Error on line: " << lineNumber << ". Invalid data. Must be 10 fields. Skipping..." << endl;
        continue;
    }
    const string& size = productInformation[8];
    const string& colour = productInformation[9];
    product = make_unique <Clothing> (id, name, purchasePrice, sellingPrice, quantityBeginningInventory,
                                      quantitySold, quantityImported, size, colour);
}
else if(productType == "Cosmetic") {
    //check if product type is Cosmetic to add product
    if(productInformation.size() != 9) {
        cerr << "Error on line: " << lineNumber << ". Invalid data. Must be 9 fields. Skipping..." << endl;
        continue;
    }
    const string& typeCosmetic = productInformation[8];
    product = make_unique <Cosmetic> (id, name, purchasePrice, sellingPrice, quantityBeginningInventory,
                                      quantitySold, quantityImported, typeCosmetic);
}
else {
    cerr << "Error on line: " << lineNumber << ". Unknown product type. Skipping..." << endl;
    continue;
}
importProductFromFile(product.get()); //add product - raw pointer
```



THANK  
you

4. Att besvara frågor och redogöra för delar av min lösning