C++ KURSPROJEK

Karlskrona, 2025

Presentetion av mig och min projekt

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

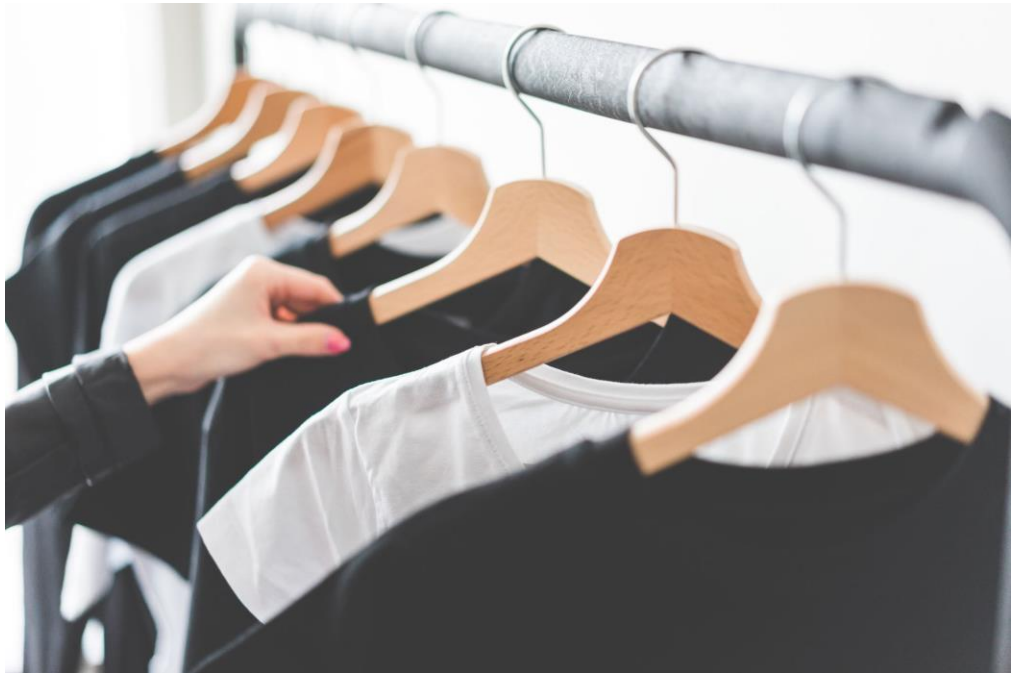
Innehållet

1. Kortfattad beskrivning av projektet och demokörning

2. Beskrivning av uppbyggnaden med utgångspunkt från klassdiagrammet

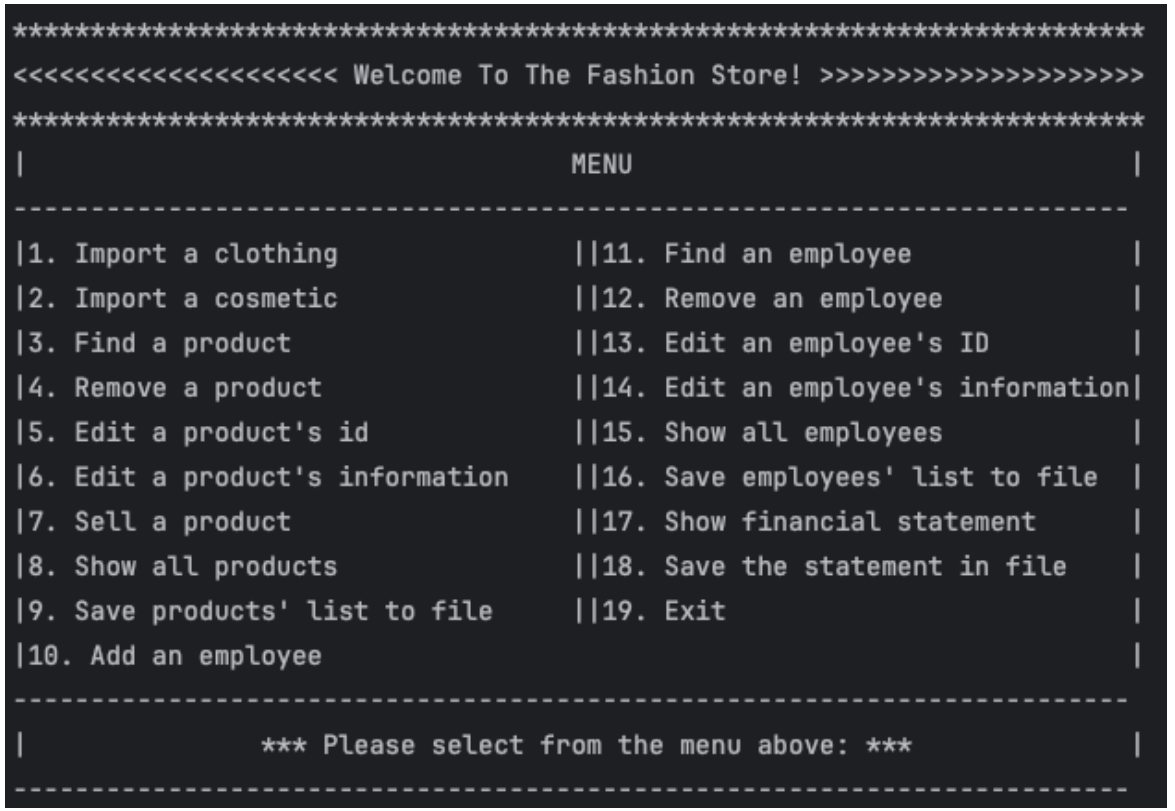
3. I koden visa att min lösning uppfyller
    - de specifika kraven
    - tilläggskrav för högre betyg

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



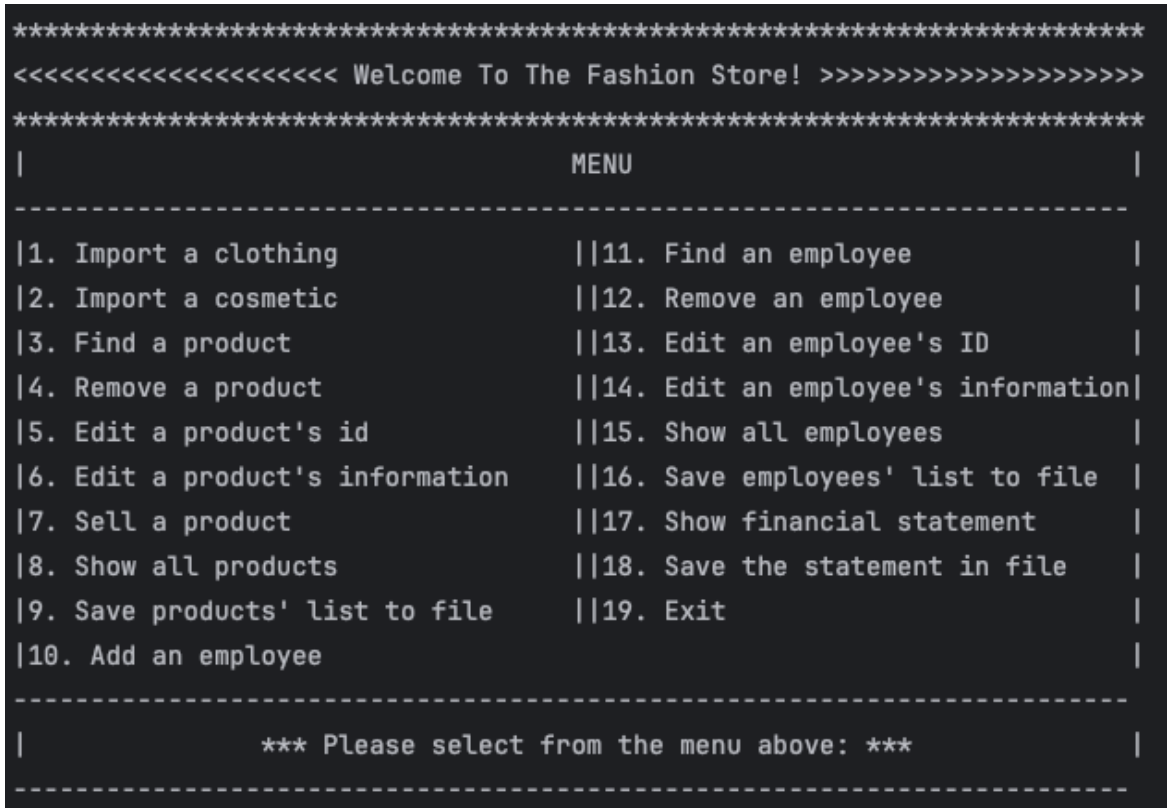
1. Kortfattad beskrivning av projektet och demokörning

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



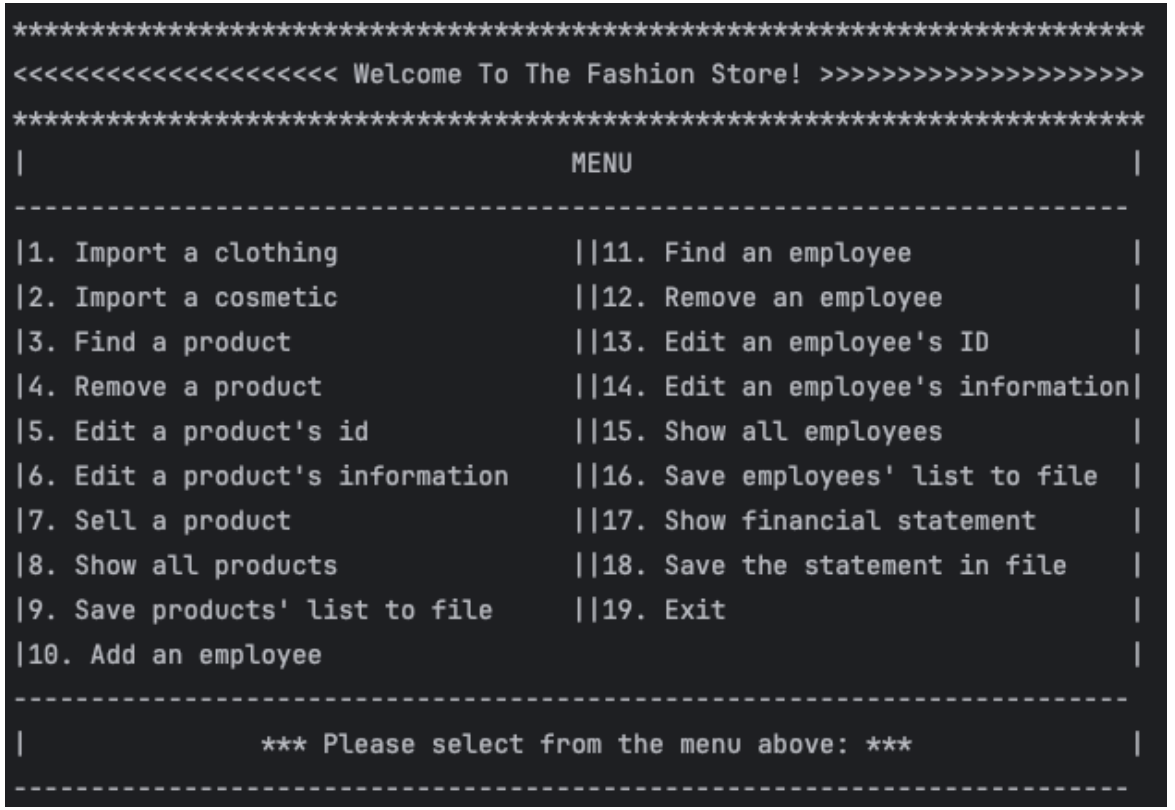
1. Kortfattad beskrivning av projektet och demokörning

- **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.



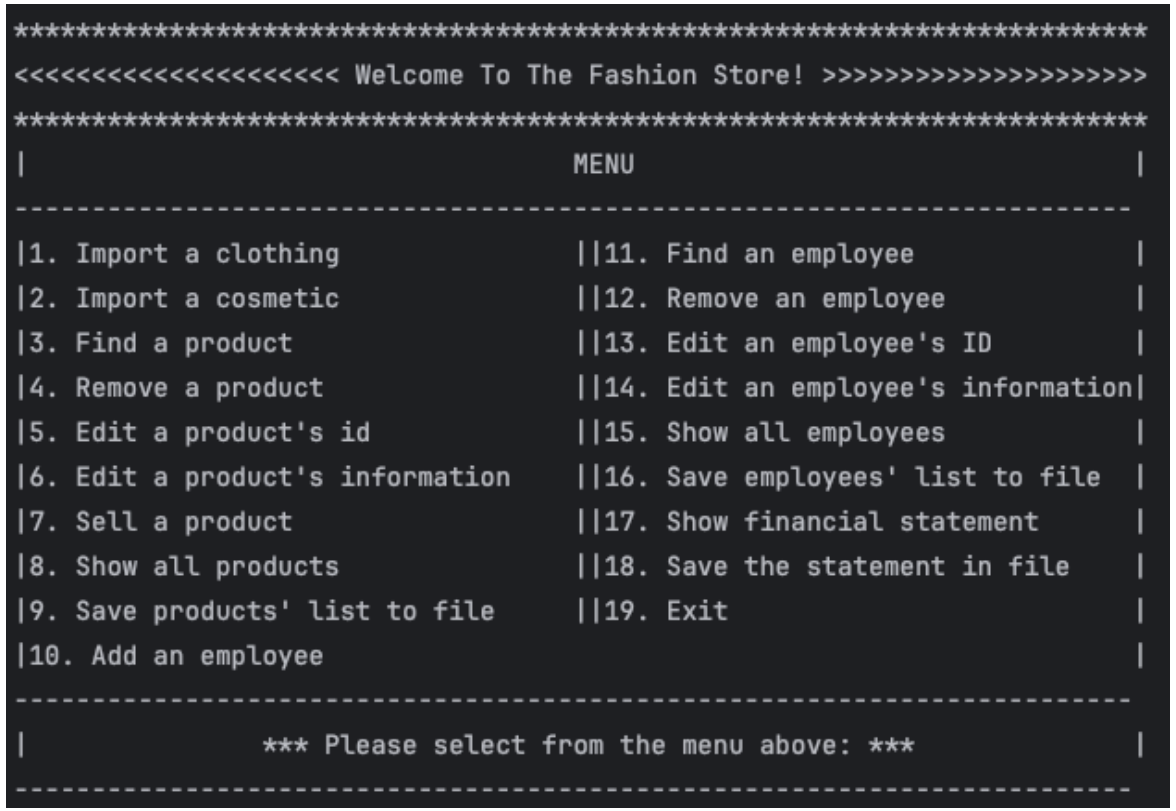
1. Kortfattad beskrivning av projektet och demokörning

- **Sälja varor:**
  - Försäljningen genomförs endast när följande villkor är uppfyllda:
    - ✓ **Rätt artikel-ID:** Användaren måste ange ett giltigt artikel-ID som finns i lager.
    - ✓ **Giltigt medarbetar-ID:** Medarbetar-ID måste vara korrekt och finnas registrerat i systemet.
    - ✓ **Tillräckligt lager:** Det måste finnas ett tillräckligt antal artiklar i lager för att genomföra försäljningen. Om lagerstatusen är för låg, ska försäljningen inte tillåtas.



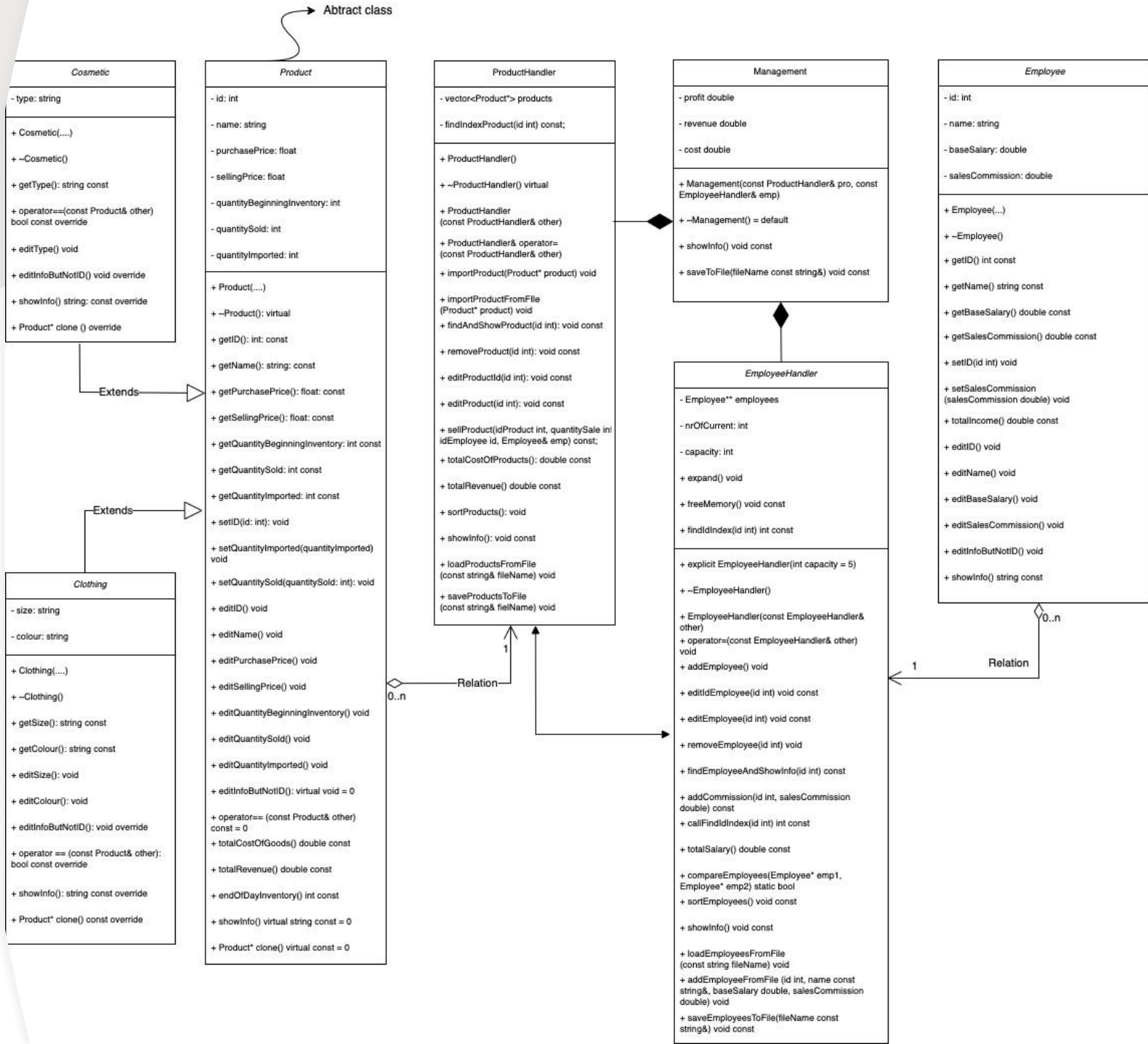
1. Kortfattad beskrivning av projektet och demokörning

- **Hitta, redigera eller ta bort med ID:**
  - Användare kan söka, redigera eller ta bort en produkt eller anställd med hjälp av dess unika 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.



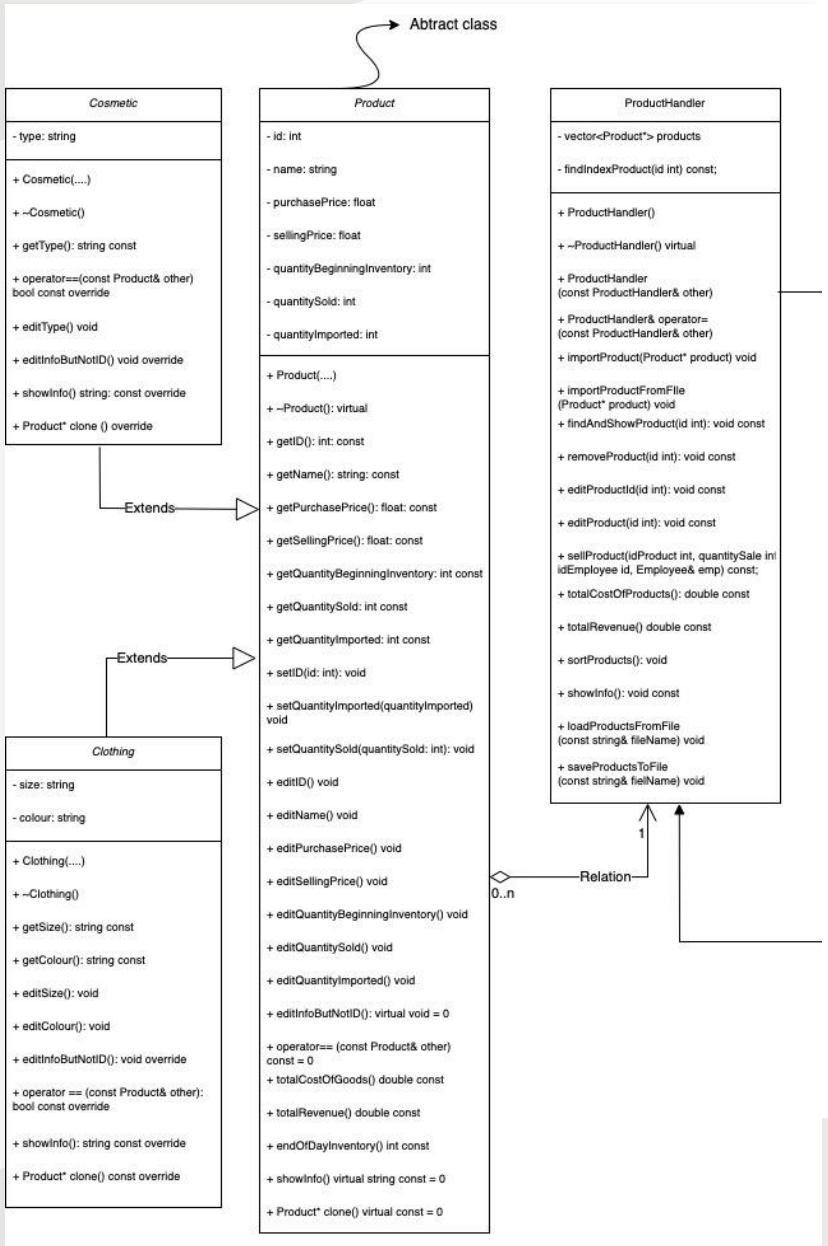
1. Kortfattad beskrivning av projektet och demokörning

2. Beskrivning av uppbyggnaden med utgångspunkt från klassdiagrammet



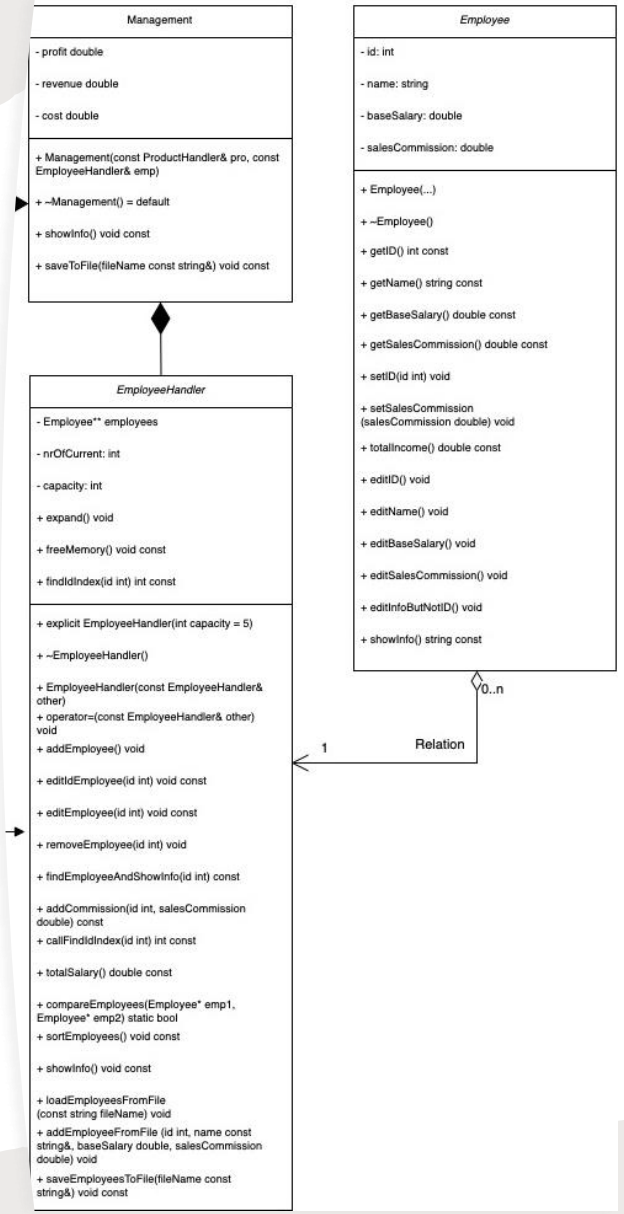
2. Beskrivning av uppbyggnaden medutgångspunkt från klassdiagrammet

```
**************************************************************************
<<<<<<<<<<<<<<<<<<<< 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: ***                 |
--------------------------------------------------------------------------
```



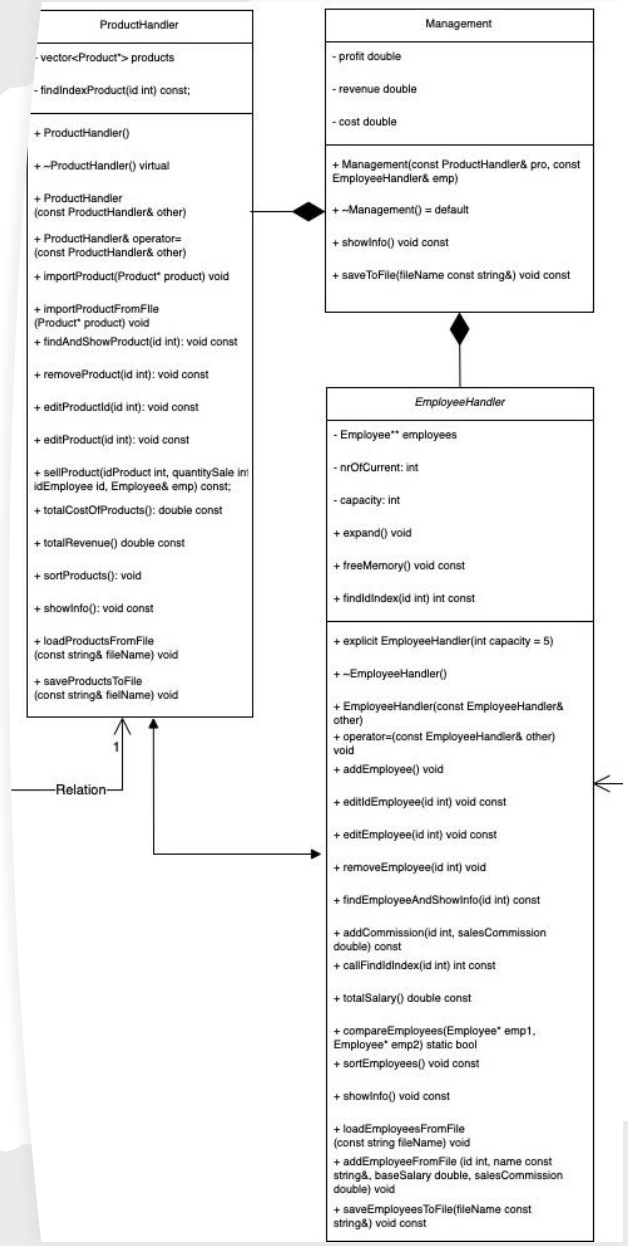
2. Beskrivning av uppbyggnaden medutgångspunkt från klassdiagrammet

```
**************************************************************************
<<<<<<<<<<<<<<<<<<<< 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: ***                 |
--------------------------------------------------------------------------
```



2. Beskrivning av uppbyggnaden medutgångspunkt 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: ***                |
--------------------------------------------------------------------------
```



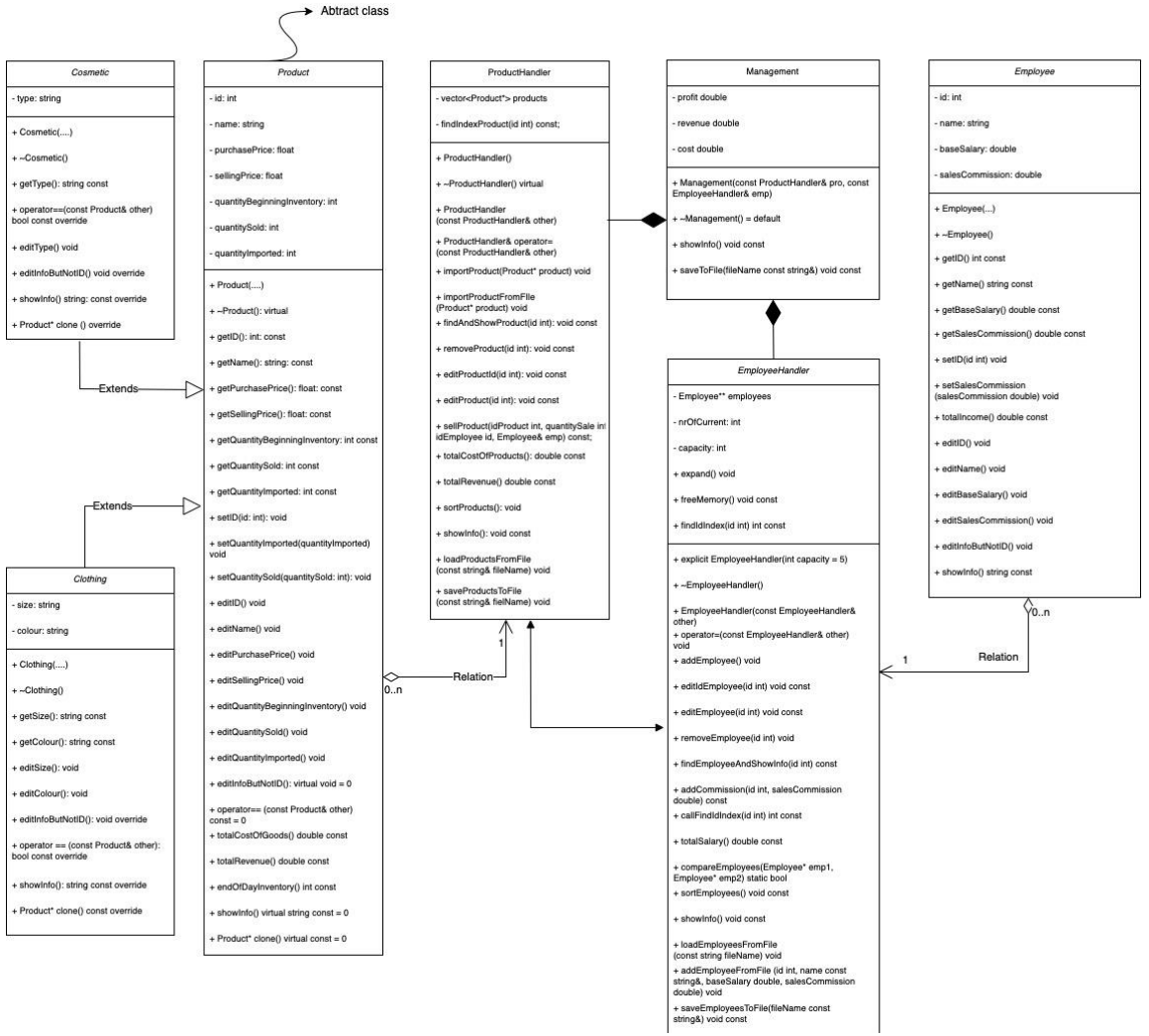
3. I koden visa att min lösning uppfyller kraven

3a. De specifika kraven

3b. Tilläggskrav för högre betyg

3a. De spectifika kraven

- ✓ Ett visst antal, minst 5, egendefinierade **relevanta** klasser



3a. De spectifika kraven

- ✓ Ett visst antal, minst 5, egendefinierade relev anta 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 spectifika kraven

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

```
class Cosmetic : public Product {
private:
    string type;
public:
    Cosmetic(int id = -1, const string& name = "?", float purc
        int quantitySold = 0, int quantityImported = 0, const st
    ~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
    int quantityBeginningInventory = 0, int
    virtual ~Product();
```

```
class Clothing : public Product{
private:
    string size;
    string colour;
public:
    Clothing(int id = -1, const string& name = "?", float pu
        int quantitySold = 0, int quantityImported = 0, const
    ~Clothing();

    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;
```

```
    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;
```

3a. De spectifika 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

```
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;
```

```
bool Product::operator==(const Product &other) const {
    return id == other.id &&
        name == other.name &&
            purchasePrice == other.purchasePrice &&
                sellingPrice == other.sellingPrice &&
                    quantityBeginningInventory == other.quantityBeginningInventory &&
                        quantitySold == other.quantitySold;
}
```

```
string Product::showInfo() const {
    ostringstream stringInfo;
    stringInfo << fixed << setprecision(n: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();
}
```

```
bool Clothing::operator==(const Product &other) const {
    const auto* otherClothing = dynamic_cast<const Clothing*>(&other);
    return otherClothing && Product::operator==(*otherClothing) &&
        size == otherClothing->size &&
            colour == otherClothing->colour;
}

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

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

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

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

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

string showInfo() const override;

Product* clone() const override;
```

```
bool Cosmetic::operator==(const Product &other) const {
    const auto* otherClothing = dynamic_cast<const Cosmetic*>(&other);
    return otherClothing && Product::operator==(*otherClothing) &&
        type == otherClothing->type;
}

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

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

3a. De spectifika kraven

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

```
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 spectifika kraven

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

```
    else {
        product->editInfoButNotID();
        products.push_back( x: 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( n: numeric_limits<streamsize>::max(),  dlm: '\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( x: product->clone());
    }
}
```

3a. De spectifika 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.loadEmployeesFromFie(fileName: "All Employees");
    productHandlerVector.loadProductsFromFie(fileName: "All Products");
```

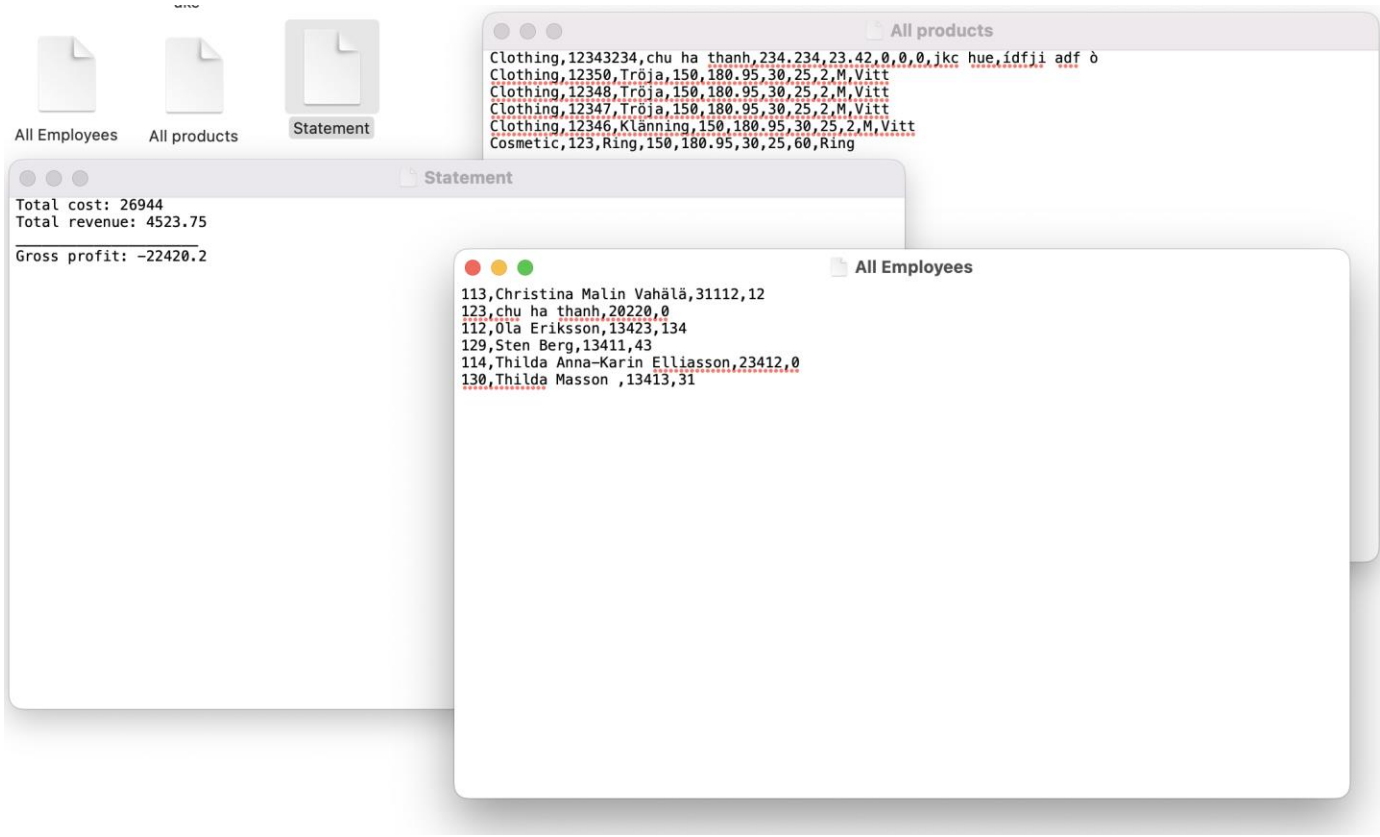
```
void sortProducts();
void showInfo () const;

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

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

void loadEmployeesFromFie(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;
```

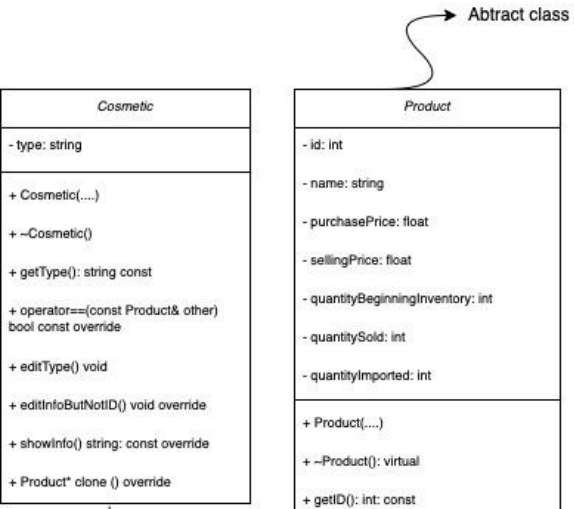
3a. De spectifika kraven

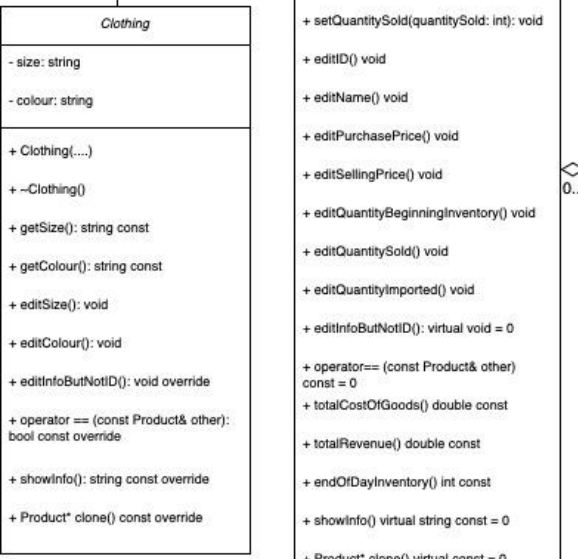
- ✓ Rimlig användning av dynamisk minnes hantering (genom användande av new och delete)
- ✓ Någon datastruktur från standardbibliot eket (vector, stack, kö, …)
- ✓ **Läsning från och/eller skrivning till tex tfiler**

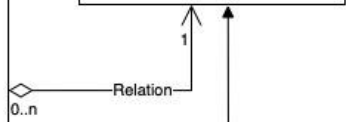


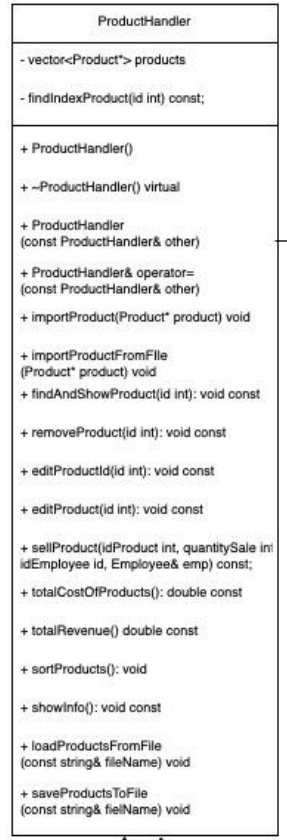
3a. De spectifika kraven

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







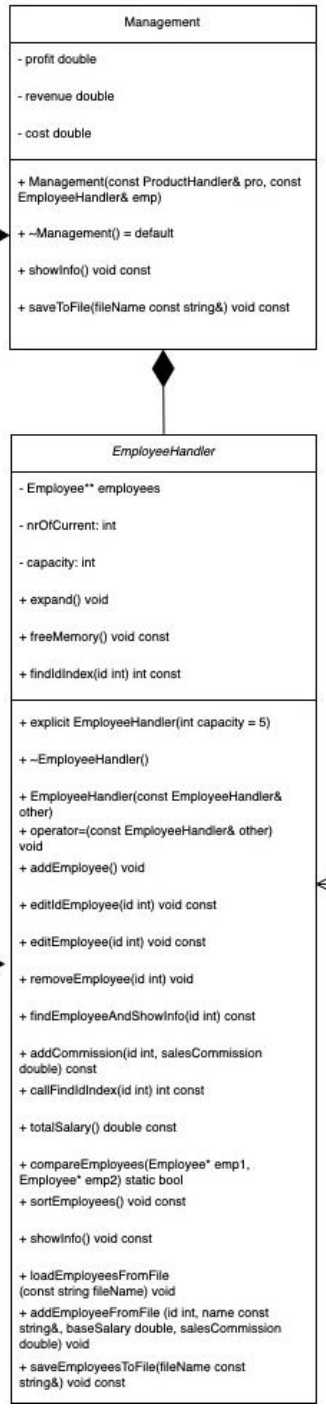



```cpp
class Cosmetic : public Product {
private:
    string type;
public:
    Cosmetic(int id = -1, const string& name = "?", float
        int quantitySold = 0, int quantityImported = 0, cons
    ~Cosmetic() override;

    string getType() const;

    void editType();
    void editInfoButNotID() override;

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

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

```cpp
class Clothing : public Product{
private:
    string size;
    string colour;
public:
    Clothing(int id = -1, const string& name = "?", float
        int quantitySold = 0, int quantityImported = 0, cons
    ~Clothing();

    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;
```

```cpp
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 = "?", flo
    int quantityBeginningInventory = 0, int quantitySold
    virtual ~Product();

    int getID() 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;

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

```cpp
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 loadProductsFromFie(const string& fileName);
    void saveProductsToFile(const string& fileName) const;

};
```




```cpp
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;
};
```

```cpp
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);

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

    void loadEmployeesFromFie(const string& fileName);
    void addEmployeeFromFile(int id, const string& name, double ba
    void saveEmployeesToFile(const string& fileName) const;
```

```cpp
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;
};
```

3a. De spectifika kraven

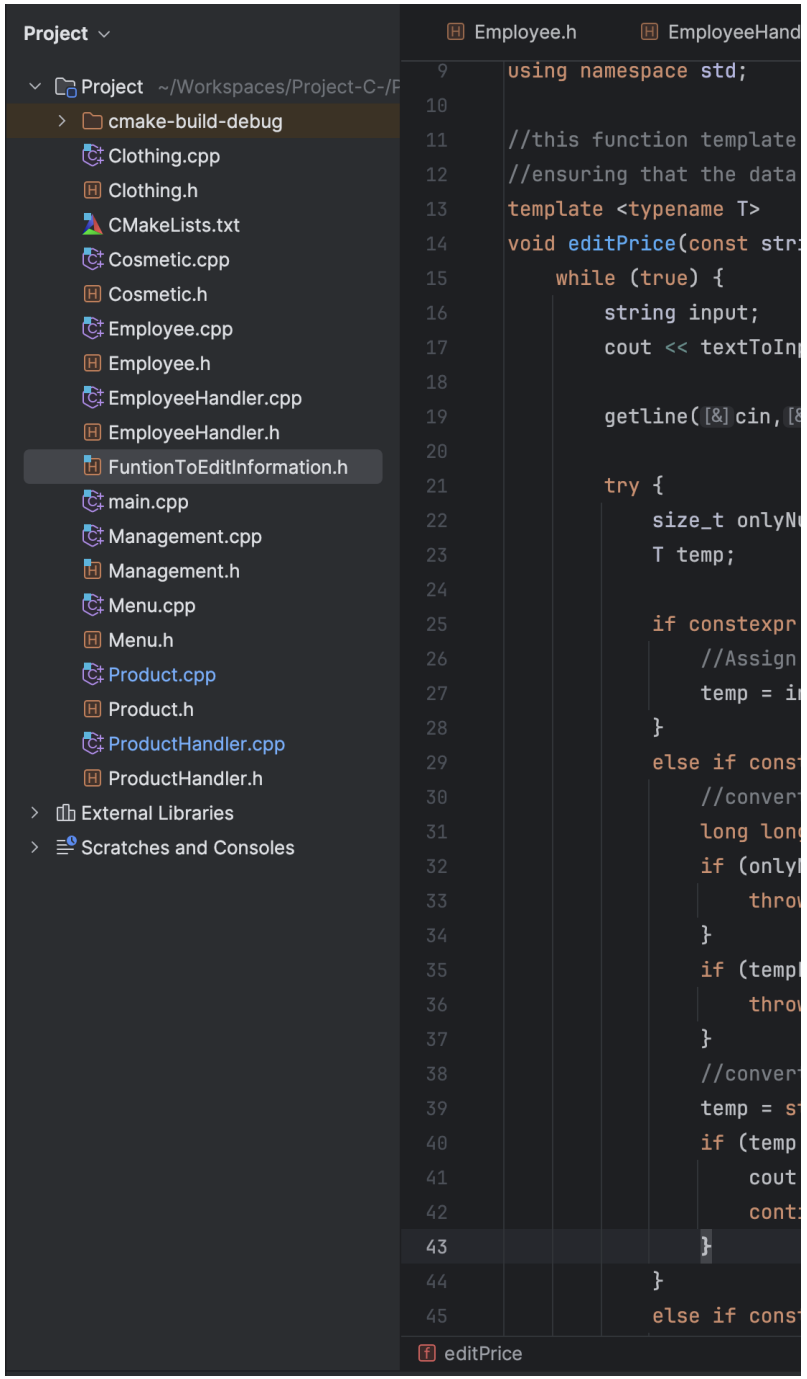
- ✓ Koden ska överenstämma med klassdiagrammet, dvs de relationer som finns i klassdiagrammet ska vara implementerade
- ✓ I normalfallet ska alla medlemsvariabler vara privata.
- ✓ **Koden ska vara "städad", dvs det ska inte finnas utkommenterade kodavsnitt.**

Tillägg bilden när jag kör prodgrammet i labbet. Går inte idag för att de har tentor hela tiden

3a. De spectifika kraven

- ✓ Koden ska överenstämma 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
    EmployeeHandler& operator=(const Empl

    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;
```




```
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& errorM
float validateAndConvertToFloat(const string& str, const string& er
double validateAndConvertToDouble(const string& str, const string&


#endif //MENU_H
```

3a. De spectifika kraven

- ✓ Koden ska överenstämma 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& 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 loadEmployeesFromFie(const string& fileName);
```

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

    string getType() const;

    void editType();
    void editInfoButNotID() override;

    bool operator==(const Product &other) const ov

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

```
    double profit;
    double revenue;
    double cost;

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

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

```
private:
    string size;
    string colour;
public:
    Clothing(int id = -1, const string& name = "?", float p
        int quantitySold = 0, int quantityImported = 0, const
    ~Clothing();

    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;
```

```
    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 spectifika kraven

- ✓ Koden ska överenstämma 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;
    }
}
```

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

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

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

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

```
~Employee();
```

```
~Management();
```

3a. De spectifika kraven

- ✓ Koden ska överenstämma 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..." << en
    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: productInfor
        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: productInformatio
        errorMessage: "Imported quantity must be an integer");

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

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

	C	D	E	F	G	
	.a inte	hitta inte	x			
	.ck	gick	x			
	skriva ut meddelande att information inte ändras. kom till menyval	skriva ut meddelande att information inte ändras. kom till menyval men det räkna \n till menyval och visat fel meddelande		x	sakna cin.ignore(..)	kolla o likadan
		Check menu 14				
	gick att kolla	gick att kolla	x			
	gick inte	gick inte	x			
	gick inte	gick inte	x			
	gick inte	gick inte	x			
	gick inte	gick inte	x			
	gick inte	gick inte	x			
	gick inte	gick inte	x			
	gick inte	gick inte	x			
	,edan)	hitta	x			
	s inte)	hitta inte	x			
	ändra id	gick	x			
		skriva ut meddelande att information inte ändras. kom till menyval	x			
	ör att ändra andra mation	gick	x			
	ɔ"	skriva ut meddelande att information inte ändras. kom till menyval	x			
		Check menu 15				
		show all employee after sort	x			
		Check menu 16				
		save all empoyee after sort				
		Check menu 17				
		calculate and show the statement				
		Check menu 18				
		save the statement				
		Check menu 19				
86		exit				
87						

	C	D	E	F	G	H
	Expected results	**Actual result**	**Pass**	**Fail**	**Reason**	**Fix**
		Check menu				
	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! Error: The input data is not valid!Please try again! Error: The input data is not valid!Please try again! 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	gick inte	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	gick -> exit	kom till menyval 19, exit	x			
	1	menu 1	kom till menyval 1, insert a product	x			
		yes/no question				
1	yes	ok	ok	x			
2	no	ok	ok	x			
3	YES	ok	ok	x			
4	NO	ok	ok	x			
5	YeS	ok	ok	x			

3b. Tilläggskrav för högre betyg

- ✓ Relevant användande av funktionspekare

```
bool EmployeeHandler::compareEmployees(Employee *emp1, Employee *emp2) {
    //Convert all names to lowercase for comparison
    string nameEmp1 = emp1->getName();
    string nameEmp2 = emp2->getName();
    transform(first: nameEmp1.begin(), last: nameEmp1.end(), result: nameEmp1.begin(), ::
    transform(first: nameEmp2.begin(), last: nameEmp2.end(), result: nameEmp2.begin(), ::

    //Use string stream to split name into words
    istringstream nameStream1(nameEmp1);
    istringstream nameStream2(nameEmp2);

    string word1, word2;
    //Compare every word in name
    while (nameStream1 >> word1 && nameStream2 >> word2) {
        if (word1 != word2) {
            return word1 < word2;  //compare if the words are different.w
        }
    }

    //if all words are same then sort by ID in descending order
    return (emp1->getId()) > (emp2->getId());
}


void EmployeeHandler::sortEmployees() const {
    sort(first: employees, last: employees + nrOfCurrent, compareEmployees);
}
```

3b. Tilläggskrav för högre betyg

- ✓ Relevant användande av funkti onspekare
- ✓ **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. Tilläggskrav för högre betyg

- ✓ Relevant användande av funktionspekare
- ✓ 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 = validateAndConvertToDouble( str: employeeInformation[2],
        errorMessage: "Base salary must be a number");
    double salesCommission = validateAndConvertToDouble( 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 funktionspekare
- ✓ Relevant användande av Lamdautryck
- ✓ Användande av undantag-shantering 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
```

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