Laborator POO - PC Builder

5/11/2024

Introducere

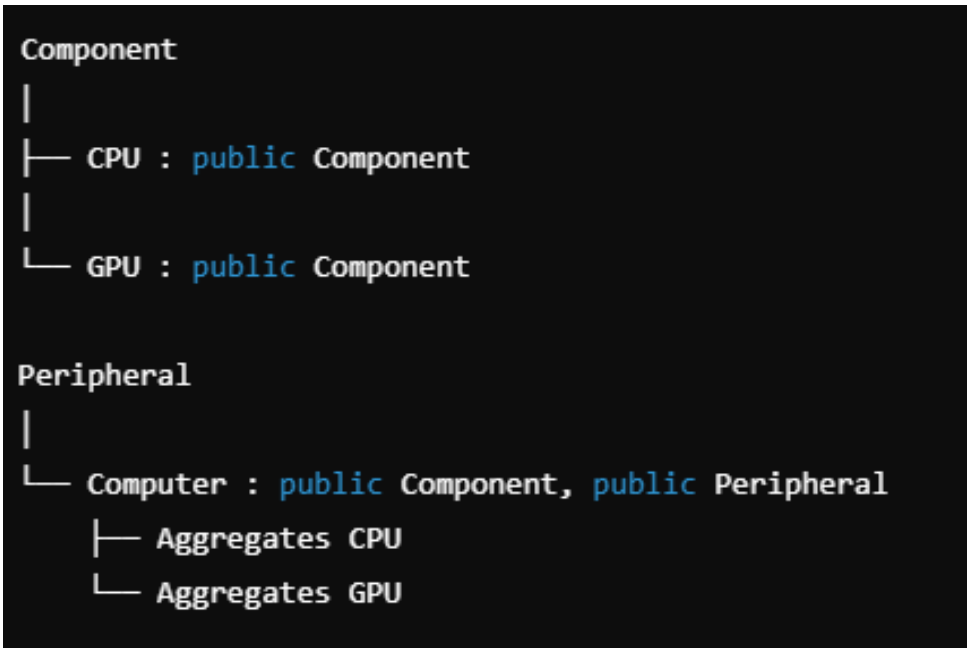
Acest laborator presupune implementarea unei structuri de clasă pentru un proiect de construcție a unui PC, cu scopul de a înțelege și utiliza moștenirea în C++ și conceptele de alocare dinamică.

Ierarhia Claselor

Ierarhia claselor pentru proiectul de construcție a unui PC este următoarea:

- **Component**
  - Clasa de bază pentru componentele interne ale unui computer.
  - Atribute: `name` și `manufacturer`.
  - Clase derivate:
    - **CPU : public Component** - Conține atributele `cores` și `frequency`.
    - **GPU : public Component** - Conține atributul `memory`.
- **Peripheral**
  - Reprezintă dispozitivele periferice.
  - Atribute: `name` și `manufacturer`.
- **Computer : public Component, public Peripheral**
  - Moștenire multiplă din `Component` și `Peripheral`.
  - Conține obiecte `CPU` și `GPU` prin agregare.

Vizualizare Grafică a Ierarhiei Claselor



Codul sursă

Mai jos este structura codului sursă. Unde este indicat, completați metodele necesare.

Component.hpp

```
#ifndef COMPONENT_HPP
#define COMPONENT_HPP

#pragma warning (disable : 4996) // strict pentru VS pentru a da disable la
    warning-urile generate de 'strcpy'

#include <iostream>
#include <cstring>

class Component {
protected:
    char* name;
    char* manufacturer;

public:
    Component(const char* name, const char* manufacturer);
    Component(const Component& other);
    Component& operator=(const Component& other);
    ~Component();

    void displayInfo() const;

    friend std::ostream& operator<<(std::ostream& out, const Component& comp);
    friend std::istream& operator>>(std::istream& in, Component& comp);
};

#endif // COMPONENT_HPP
```

Component.cpp

```cpp
#include "Component.hpp"

Component::Component(const char* name, const char* manufacturer) {
    // TODO: Implementare constructor
}

Component::Component(const Component& other) {
    // TODO: Implementare constructor de copiere
}

Component& Component::operator=(const Component& other) {
    // TODO: Implementare operator de atribuire
    return *this;
}

Component::~Component() {
    // TODO: Implementare destructor
}

void Component::displayInfo() const {
    // TODO: Implementare metoda displayInfo
}

std::ostream& operator<<(std::ostream& out, const Component& comp) {
    // TODO: Implementare operator <<
    return out;
}

std::istream& operator>>(std::istream& in, Component& comp) {
    // TODO: Implementare operator >>
    return in;
}
```

CPU.hpp

```cpp
#ifndef CPU_HPP
#define CPU_HPP

#pragma warning (disable : 4996)

#include <iostream>
#include "Component.h"

class CPU : public Component {
private:
    int cores;
    float frequency; // GHz

public:
    CPU(const char* name, const char* manufacturer, int cores, float frequency)
        ;
    void displayInfo() const;

    friend std::ostream& operator<<(std::ostream& out, const CPU& cpu);
    friend std::istream& operator>>(std::istream& in, CPU& cpu);
};

#endif // CPU_HPP
```

CPU.cpp

```cpp
#include "CPU.hpp"

CPU::CPU(const char* name, const char* manufacturer, int cores, float frequency
    ) {
    // TODO: Implementare constructor
}

void CPU::displayInfo() const {
    // TODO: Implementare metoda displayInfo
}

std::ostream& operator<<(std::ostream& out, const CPU& cpu) {
    // TODO: Implementare operator <<
    return out;
}

std::istream& operator>>(std::istream& in, CPU& cpu) {
    // TODO: Implementare operator >>
    return in;
}
```

GPU.hpp

```cpp
#ifndef GPU_HPP
#define GPU_HPP

#pragma warning (disable : 4996)

#include <iostream>
#include "Component.h"

class GPU : public Component {
private:
    int memory; // GB

public:
    GPU(const char* name, const char* manufacturer, int memory);
    void displayInfo() const;

    friend std::ostream& operator<<(std::ostream& out, const GPU& gpu);
    friend std::istream& operator>>(std::istream& in, GPU& gpu);
};

#endif // GPU_HPP
```

GPU.cpp

```cpp
#include "GPU.hpp"

GPU::GPU(const char* name, const char* manufacturer, int memory) {
    // TODO: Implementare constructor
}

void GPU::displayInfo() const {
    // TODO: Implementare metoda displayInfo
}

std::ostream& operator<<(std::ostream& out, const GPU& gpu) {
    // TODO: Implementare operator <<
    return out;
}

std::istream& operator>>(std::istream& in, GPU& gpu) {
    // TODO: Implementare operator >>
    return in;
}
```

Peripheral.hpp

```cpp
#ifndef PERIPHERAL_HPP
#define PERIPHERAL_HPP

#pragma warning (disable : 4996)

#include <iostream>
#include <cstring>

class Peripheral {
protected:
    char* name;
    char* manufacturer;

public:
    Peripheral(const char* name, const char* manufacturer);
    Peripheral(const Peripheral& other);
    Peripheral& operator=(const Peripheral& other);
    ~Peripheral();

    void displayInfo() const;

    friend std::ostream& operator<<(std::ostream& out, const Peripheral&
        peripheral);
    friend std::istream& operator>>(std::istream& in, Peripheral& peripheral);
};

#endif // PERIPHERAL_HPP
```

Peripheral.cpp

```cpp
#include "Peripheral.h"

Peripheral::Peripheral(const char* name, const char* manufacturer) {
    // TODO: Implementare constructor
}

Peripheral::Peripheral(const Peripheral& other) {
    // TODO: Implementare constructor de copiere
}

Peripheral& Peripheral::operator=(const Peripheral& other) {
    // TODO: Implementare operator de atribuire
    return *this;
}

Peripheral::~Peripheral() {
    // TODO: Implementare destructor
}

void Peripheral::displayInfo() const {
    // TODO: Implementare metoda displayInfo
}

std::ostream& operator<<(std::ostream& out, const Peripheral& peripheral) {
    // TODO: Implementare operator <<
    return out;
}

std::istream& operator>>(std::istream& in, Peripheral& peripheral) {
    // TODO: Implementare operator >>
    return in;
}
```

Computer.hpp

```cpp
#ifndef COMPUTER_HPP
#define COMPUTER_HPP

#include <iostream>
#include "Component.hpp"
#include "Peripheral.hpp"
#include "CPU.h"
#include "GPU.h"

class Computer : public Component, public Peripheral {
private:
    CPU cpu;
    GPU gpu;

public:
    Computer(const char* name_comp, const char* manufacturer_comp, const char*
        name_per, const char* manufacturer_per, const CPU& cpu, const GPU& gpu);
    void displayComputerInfo() const;
};

#endif // COMPUTER_HPP
```

Computer.cpp

```cpp
#include "Computer.hpp"

Computer::Computer(const char* name_comp, const char* manufacturer_comp, const
    char* name_per, const char* manufacturer_per, const CPU& cpu, const GPU& gpu
    ) {
    // TODO: Implementare constructor
}

void Computer::displayComputerInfo() const {
    // TODO: Implementare metoda displayComputerInfo
}
```

Main.cpp

```cpp
#include <iostream>
#include "Computer.hpp"
#include "CPU.hpp"
#include "GPU.hpp"

int main() {
    const int numComputers = 2;
    Computer* computers[numComputers];

    // TODO: Initializati obiectele CPU si GPU
    // TODO: Creati obiectele Computer in vectorul computers

    std::cout << "Computer Configurations:\n\n";
    for (int i = 0; i < numComputers; ++i) {
        computers[i]->displayComputerInfo();
        std::cout << "----------------------" << std::endl;
    }

    // TODO: Eliberati memoria alocata
    return 0;
}
```

Punctaj:

- Implementare clase - 5p
- Implementare main - 3p
- Funcționalitate - 1p
- Oficiu - 1p