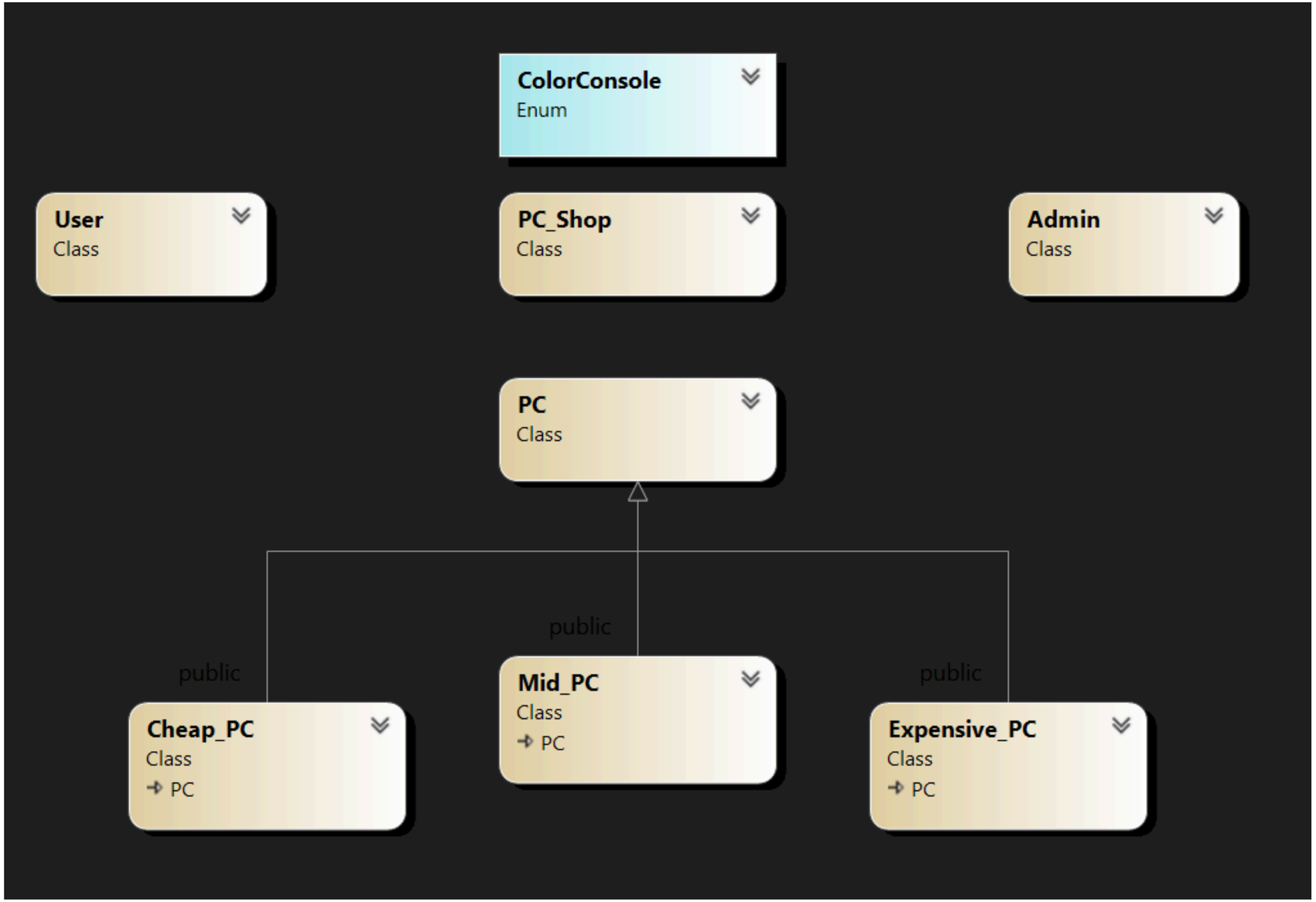
C++ curs final Project

For STEP ACADEMY

Class Diagram



Constructors
Setters
Getters

```
class PC {
protected:
    int id;
    string name;
    string description;
    int cost;
public:
    static int counter;

    PC() { id = counter; name = "Undefined"; description = "None"; cost = 0; }
    PC(int id, string name, string description, int cost) { ... }

    void setName(string name) { this->name = name; }
    void setDescription(string descriprion) { this->description = description; }
    void setCost(int) { this->cost = cost; }

    string getName()const { return name; }
    string getDescription()const { return description; }
    int getCost()const { return cost; }

    int getID() const { return id; }
    virtual void showInfo() const = 0;

    virtual string getInfo() const { ... }

    virtual string getAllPropertiesAsString()const = 0;


};
```

Child class

```
class Cheap_PC:public PC {
protected:
    bool IntegratedGPU;
    float SALE; //*0.75 = 15 %
public:
    Cheap_PC() :PC() { IntegratedGPU = true; SALE = 1; }
    Cheap_PC(int id ,string name, string description, bool IntegratedGPU, float SALE ,
    {
        this->IntegratedGPU = IntegratedGPU; this->SALE = SALE;
    }

    void setGPU(bool IntegratedGPU) { this->IntegratedGPU = IntegratedGPU; }
    bool getGPU()const { return IntegratedGPU; }

    void setSALE(float SALE) { this->SALE = SALE; }
    float getSALE()const { return SALE; }

    void showInfo()const override;


    string getInfo() const override { ... }

    string getAllPropertiesAsString()const override;
    void setAllPropertiesFromString(string properties);
};
```

Overloaded Operator+

```
void addPC(PC* pc) {
    *this + pc;
}
```

```
class PC_Shop
{
    deque<PC*> sold_pcs;
    static const int MAX_SOLD_PCS = 10;

    string name;
    vector<PC*> pcs;
public:
    PC_Shop() { name = "Undefined"; }
    PC_Shop(string name) { this->name = name; }

    ~PC_Shop() { ... }
    PC_Shop& operator+(PC* pc) {
        pcs.push_back(pc);
        writePCListToFile();
        return *this;
    }
```

Static counter

For create unique ID

```
class PC {
protected:
    int id;
    string name;
    string description;
    int cost;
public:
    static int counter;

    PC() { id = counter; name = "Undefined"; description = "None"; cost = 0; }
    PC(int id, string name, string description, int cost) { ... }

    void setName(string name) { this->name = name; }
    void setDescription(string descriprion) { this->description = description; }
    void setCost(int) { this->cost = cost; }
```

STL vector and beque

```
vector<User> users;
vector<Admin> admins;
bool DONE = false;
```

beque for 10 limit obj list of saled pc

```
class PC_Shop
{
    deque<PC*> sold_pcs;
    static const int MAX_SOLD_PCS = 10;

    string name;
    vector<PC*> pcs;
public:
    PC_Shop() { name = "Undefined"; }
    PC_Shop(string name) { this->name = name; }

    ~PC_Shop() { ... }
    PC_Shop& operator+(PC* pc) {
        pcs.push_back(pc);
        writePCListToFile();
        return *this;
    }
```

Add , Del , find

```
PC* findPCByName(string pcName) {
    for (PC* pc : pcs) {
        if (pc->getName() == pcName) {
            return pc;
        }
    }
    return nullptr;
}

void removePCFromVector(vector<PC*>& vec, PC* pc) { ... }
```

```
PC* findPCByID(int pcID) {
    for (PC* pc : pcs) {
        if (pc->getID() == pcID)
            return pc;
        }
    }
    return nullptr;
}
```

```
void addPC(PC* pc) {
    *this + pc;
}

void addNewCheapPC() { .

void addNewExpensivePC()

void addNewMidPC() { ...
```

Exeptions in add

```
void addNewCheapPC() {
    string name;
    string description;
    bool integratedGPU;
    float sale;
    int cost;

    try {
        cout << "Введите название компьютера: ";
        cin >> name;
        cout << "Введите описание компьютера: ";
        cin.ignore();
        getline(cin, description);

        cout << "Есть ли интегрированный GPU? (1 - да, 0 - нет): ";
        cin >> integratedGPU;
        if (integratedGPU != 1 && integratedGPU != 0) {
            throw runtime_error("Wrong input for GPU");
        }

        cout << "Введите скидку: ";
        cin >> sale;
        if (sale > 1 || sale <= 0) {
            throw runtime_error("Wrong Sale");
        }

        cout << "Введите стоимость компьютера: ";
        cin >> cost;
        if (cost <= 0) {
            throw runtime_error("Wrong cost");
        }

        Cheap_PC* new_cheap_pc = new Cheap_PC(PC::counter++, name, description, integratedGPU, sale, cost);
        addPC(new_cheap_pc);
    }
    catch (const runtime_error& e) {
        SetRed();
        cout << e.what() << endl;
        SetBlue();
    }
}
```

Work with files

```
private:
    void readCounterFromFile() { ... }

    void writeCounterToFile() const { ... }

    void readPCListFromFile() { ... }

    void writePCListToFile() const { ... }


    void writeSoldPCsToFile(const string& userName) const { ... }

    void readSoldPCsFromFile() { ... }




    void clearFile() { ... }

};
```

Sorted output

```
void sortPCsByName(vector<PC*> tempPCs) {
    sort(tempPCs.begin(), tempPCs.end(), [](PC*a , PC * b){ return a->getName() < b->getName();
    for (PC* pc : tempPCs) {
        SetRed();
        cout << "ID: " << pc->getID();
        cout << endl;
        SetBlue();
        pc->showInfo();
        cout << endl;
    }
    cin.ignore();
    cin.get();
}

void sortPCsByCost(vector<PC*> tempPCs) {
    sort(tempPCs.begin(), tempPCs.end(),[](PC*a , PC* b){ return a->getCost() < b->getCost(); })
    for (PC* pc : tempPCs) {
        SetRed();
        cout << "ID: " << pc->getID();
        cout << endl;
        SetBlue();
        pc->showInfo();
        cout << endl;
    }
    cin.ignore();
    cin.get();
}
```

Main

```cpp
int main() {
    thread th(animation);
    thread thr1(loadUsers);
    thread thr2(loadAdmins);

    Sleep(3300);
    DONE = true;

    th.detach();
    thr1.detach();
    thr2.detach();
    Sleep(200);

    cout << "Users: \n" <<endl;
    Main_Menu();
    return 0;
}
```

End.