computerTechnology.hpp

#pragma once

#include <string>

class ComputerTechnology

{

protected:

    std::string Manufacturer;

public:

    ComputerTechnology(std::string Manufacturer = "DEXP");

    virtual void Print() const noexcept;

    const std::string& GetManufacturer() const noexcept;

    void SetManufacturer(std::string manufacturer);

};

computerTechnology.cpp

#include "computerTechnology.hpp"

#include <iostream>

ComputerTechnology::ComputerTechnology(std::string manufacturer) : Manufacturer{manufacturer}

{ }

void ComputerTechnology::Print() const noexcept

{

std::cout << "Manufacturer: " << Manufacturer << std::endl;

}

const std::string& ComputerTechnology::GetManufacturer() const noexcept

{

return Manufacturer;

}

void ComputerTechnology::SetManufacturer(std::string manufacturer)

{

Manufacturer = manufacturer;

}

monitor.hpp

#pragma once

#include "computerTechnology.hpp"

#include <string>

#include <vector>

class Monitor : public ComputerTechnology

{

    std::vector<int> Resolution;

    float ScreenSize;

    int RefreshRate;

public:

    Monitor(std::string manufacturer = "Samsung", std::vector<int> resolution = {1920, 1080},

        float screenSize = 23.8, int refreshRate = 60);

    void Print() const noexcept override;

    const std::string& GetManufacturer() const noexcept;

    void SetManufacturer(std::string manufacturer);

    const std::vector<int>& GetResolution() const noexcept;

    void SetResolution(std::vector<int> resolution);

    void SetResolution(int width, int height);

    float GetScreenSize() const noexcept;

    void SetScreenSize(float screenSize);

    int GetRefreshRate() const noexcept;

    void SetRefreshRate(int refreshRate);

};

monitor.cpp

#include "monitor.hpp"

#include <iostream>

#include <stdexcept>

Monitor::Monitor(std::string manufacturer, std::vector<int> resolution, float screenSize, int refreshRate) :

    ComputerTechnology{manufacturer}, Resolution{resolution}, ScreenSize{screenSize}, RefreshRate{refreshRate}

{

    if (resolution.size() != 2 || resolution[0] < 1 || resolution[1] < 1)

        throw std::invalid\_argument("Resolution must be a vector of two positive integers");

    if (screenSize <= 0)

        throw std::invalid\_argument("Screen size must be a positive float number");

    if (refreshRate < 1)

        throw std::invalid\_argument("Refresh rate must be a positive integer");

}

void Monitor::Print() const noexcept

{

    std::cout << "Manufacturer: " << Manufacturer << ", resolution: " << Resolution[0] << 'x' << Resolution[1]

        << ", screen size: " << ScreenSize << "\", refresh rate: " << RefreshRate << " Hz" << std::endl;

}

const std::string& Monitor::GetManufacturer() const noexcept

{

    return Manufacturer;

}

void Monitor::SetManufacturer(std::string manufacturer)

{

    Manufacturer = manufacturer;

}

const std::vector<int>& Monitor::GetResolution() const noexcept

{

    return Resolution;

}

void Monitor::SetResolution(std::vector<int> resolution)

{

    if (resolution.size() != 2 || resolution[0] < 1 || resolution[1] < 1)

        throw std::invalid\_argument("Resolution must be a vector of two positive integers");

    Resolution = resolution;

}

void Monitor::SetResolution(int width, int height)

{

    if (width < 1 || height < 1)

        throw std::invalid\_argument("Resolution consists of two positive integers");

    Resolution = std::vector<int>({width, height});

}

float Monitor::GetScreenSize() const noexcept

{

    return ScreenSize;

}

void Monitor::SetScreenSize(float screenSize)

{

    if (screenSize <= 0)

        throw std::invalid\_argument("Screen size must be a positive float number");

    ScreenSize = screenSize;

}

int Monitor::GetRefreshRate() const noexcept

{

    return RefreshRate;

}

void Monitor::SetRefreshRate(int refreshRate)

{

    if (refreshRate < 1)

        throw std::invalid\_argument("Refresh rate must be a positive integer");

    RefreshRate = refreshRate;

}

keyboard.hpp

#pragma once

#include "computerTechnology.hpp"

#include <string>

enum class Key { None, Membrane, Mechanical, Optical };

enum class Connection { None, Wired, Wireless };

class Keyboard : public ComputerTechnology

{

    std::string Color;

    Key KeysType;

    Connection ConnectionType;

    bool HasBacklight;

public:

    Keyboard(std::string manufacturer = "Logitech", std::string color = "Black",

        Key keysType = Key::None, Connection connectionType = Connection::None, bool hasBacklight = false);

    Keyboard(std::string manufacturer = "Logitech", std::string color = "Black",

        std::string keysType = "none", std::string connectionType = "none", bool hasBacklight = false);

    void Print() const noexcept override;

    const std::string& GetManufacturer() const noexcept;

    void SetManufacturer(std::string manufacturer);

    const std::string& GetColor() const noexcept;

    void SetColor(std::string color);

    Key GetKeysType() const noexcept;

    void SetKeysType(Key keysType);

    Connection GetConnectionType() const noexcept;

    void SetConnectionType(Connection connectionType);

    bool GetHasBacklight() const noexcept;

    void SetHasBacklight(bool hasBacklight);

};

keyboard.cpp

#include "keyboard.hpp"

#include <iostream>

#include <cctype>

Keyboard::Keyboard(std::string manufacturer, std::string color, Key keysType, Connection connectionType, bool hasBacklight) :

    ComputerTechnology{manufacturer}, Color{color}, KeysType{keysType}, ConnectionType{connectionType}, HasBacklight{hasBacklight}

{

    for (auto& character : color)

        if (!std::isalpha(character))

            throw std::invalid\_argument("Color can't consist of anything other than letters");

}

Keyboard::Keyboard(std::string manufacturer, std::string color, std::string keysType, std::string connectionType,

    bool hasBacklight) : Keyboard{manufacturer, color, Key::None, Connection::None, hasBacklight}

{

    if (keysType == "membrane")

        KeysType = Key::Membrane;

    else if (keysType == "mechanical")

        KeysType = Key::Mechanical;

    else if (keysType == "optical")

        KeysType = Key::Optical;

    else

        KeysType = Key::None;

    if (connectionType == "wired")

        ConnectionType = Connection::Wired;

    else if (connectionType == "wireless")

        ConnectionType = Connection::Wireless;

    else

        ConnectionType = Connection::None;

}

void Keyboard::Print() const noexcept

{

    std::string KeysTypeLiterals[] = { "none", "membrane", "mechanical", "optical" };

    std::string ConnectionTypeLiterals[] = { "none", "wired", "wireless" };

    std::cout << "Manufacturer: " << Manufacturer << ", color: " << Color

        << ", type of keys: " << KeysTypeLiterals[static\_cast<int>(KeysType)]

        << ", connection type: " << ConnectionTypeLiterals[static\_cast<int>(ConnectionType)]

        << ", has lighting: " << std::boolalpha << HasBacklight << std::noboolalpha << std::endl;

}

const std::string& Keyboard::GetManufacturer() const noexcept

{

    return Manufacturer;

}

void Keyboard::SetManufacturer(std::string manufacturer)

{

    Manufacturer = manufacturer;

}

const std::string& Keyboard::GetColor() const noexcept

{

    return Color;

}

void Keyboard::SetColor(std::string color)

{

    for (auto& character : color)

        if (!std::isalpha(character))

            throw std::invalid\_argument("Color can't consist of anything other than letters");

    Color = color;

}

Key Keyboard::GetKeysType() const noexcept

{

    return KeysType;

}

void Keyboard::SetKeysType(Key keysType)

{

    KeysType = keysType;

}

Connection Keyboard::GetConnectionType() const noexcept

{

    return ConnectionType;

}

void Keyboard::SetConnectionType(Connection connectionType)

{

    ConnectionType = connectionType;

}

bool Keyboard::GetHasBacklight() const noexcept

{

    return HasBacklight;

}

void Keyboard::SetHasBacklight(bool hasBacklight)

{

    HasBacklight = hasBacklight;

}

app.cpp

#include "monitor.hpp"

#include "keyboard.hpp"

#include <iostream>

#include <list>

void main()

{

    std::list<ComputerTechnology\*>\* compTechList = new std::list<ComputerTechnology\*>();

    int input = 0;

    while (1)

    {

        std::cout << "What object do you want to make?" << std::endl << "1 - Computer technology"

            << std::endl << "2 - Monitor" << std::endl << "3 - Keyboard" << std::endl

            << "4 - Print objects" << std::endl << "5 - Exit" << std::endl;

        std::cin >> input;

        switch (input)

        {

        case 1:

        {

            std::string manufacturer;

            std::cout << "Enter manufacturer: ";

            std::cin >> manufacturer;

            compTechList->push\_back(new ComputerTechnology(manufacturer));

            break;

        }

        case 2:

        {

            std::string manufacturer;

            float screenSize;

            int width, height, refreshRate;

            std::cout << "Enter manufacturer, resolution (in format WIDTHxHEIGHT), screen size and refresh rate" << std::endl;

            std::cin >> manufacturer >> width;

            std::cin.ignore(1, 'x');

            std::cin >> height >> screenSize >> refreshRate;

            compTechList->push\_back(new Monitor(manufacturer, {width, height}, screenSize, refreshRate));

            break;

        }

        case 3:

        {

            std::string manufacturer, color, keys, connection;

            bool hasBacklight;

            std::cout << "Enter manufacturer, color, type of keys (membrane, mechanical or optical), "

                "connection type (wired or wireless) and whether there is a backlight (1 or 0)" << std::endl;

            std::cin >> manufacturer >> color >> keys >> connection >> hasBacklight;

            compTechList->push\_back(new Keyboard(manufacturer, color, keys, connection, hasBacklight));

            break;

        }

        case 4:

        {

            int i = 1;

            for (const ComputerTechnology\* item : \*compTechList)

            {

                std::cout << i++ << ": ";

                item->Print();

            }

            break;

        }

        case 5:

            for (auto item : \*compTechList)

                delete item;

            delete compTechList;

            return;

        default:

            break;

        }

    }

}