**Лабораторна робота №2**

**Перевантаження операцій.**

**Мета:** Одержати практичні навички створення абстрактних типів даних і перевантаження операцій у мові С++.

**Основний зміст роботи.**

Визначити і реалізувати клас — абстрактний тип даних. Визначити і реалізувати операції над даними цього класу. Написати і виконати програму повного тестування цього класу.

**Завдання 1**

В клас Money додати перевантаження:

* операції ++ (--): одночасно збільшує (зменшує) значення полів first і second;
* операції !: повертає значення true, якщо поле second не нульове, інакше false;
* операції бінарний +: додає до значення поля second значення скаляра;
* перетворення типу Money в string (і навпаки).

**Код програми**

Money.h

#ifndef MONEY\_H

#define MONEY\_H

#include <string>

class Money

{

private:

unsigned int \_denominations;

unsigned long \_counts;

public:

//--------- Constructors and Destructor -----------

#pragma region

Money() {}

Money(unsigned int denominations = 0, unsigned long counts = 0);

Money(const Money& other);

Money(Money&& other) noexcept;

~Money();

#pragma endregion

//--------------------------- Gets / Sets --------------------------------

#pragma region

int getDenominations() const;

long getCounts() const;

bool setDenominations(int denominations);

bool setCounts(long counts);

#pragma endregion

//------------------------ Overloaded Operators --------------------------

#pragma region

Money operator+(const Money& other) const;

Money operator-(const Money& other) const;

Money& operator=(const Money& other);

Money& operator=(Money&& other) noexcept;

bool operator==(const Money& other) const;

bool operator!=(const Money& other) const;

bool operator<(const Money& other) const;

bool operator<=(const Money& other) const;

bool operator>(const Money& other) const;

bool operator>=(const Money& other) const;

operator std::string() const;

Money& operator++(); // Prefix increment

Money operator++(int); // Postfix increment

Money& operator--(); // Prefix decrement

Money operator--(int); // Postfix decrement

friend std::ostream& operator<<(std::ostream& os, const Money& money);

friend std::istream& operator>>(std::istream& is, Money& money);

#pragma endregion

};

#endif // MONEY\_H

Money.cpp

#include "Money.h"

#include <iostream>

//---------------------- Constructors and Destructor ----------------------

#pragma region

Money::Money(unsigned int denominations, unsigned long counts)

: \_denominations(denominations), \_counts(counts)

{

std::cout << "Money::Money(int denominations, long counts) called" << std::endl;

}

Money::Money(const Money& other)

: \_denominations(other.\_denominations), \_counts(other.\_counts)

{

std::cout << "Money::Money(const Money& other) called" << std::endl;

}

Money::Money(Money&& other) noexcept

{

\_denominations = other.\_denominations;

\_counts = other.\_counts;

other.\_denominations = 0;

other.\_counts = 0;

std::cout << "Money::Money(Money&& other) called" << std::endl;

}

Money::~Money()

{

\_denominations = 0;

\_counts = 0;

std::cout << "Money::~Money called" << std::endl;

}

#pragma endregion

//--------------------------- Gets / Sets --------------------------------

#pragma region

int Money::getDenominations() const

{

return \_denominations;

}

long Money::getCounts() const

{

return \_counts;

}

bool Money::setDenominations(int denominations)

{

if (denominations >= 0)

{

\_denominations = denominations;

return true;

}

else return false;

}

bool Money::setCounts(long counts)

{

if (counts >= 0)

{

\_counts = counts;

return true;

}

else return false;

}

#pragma endregion

//------------------------ Overloaded Operators --------------------------

#pragma region

Money Money::operator+(const Money& other) const

{

std::cout << "operator+ called" << std::endl;

if (\_denominations < 0)

return Money(0, \_counts + other.\_counts);

if (\_counts < 0)

return Money(\_denominations, 0);

return Money(\_denominations + other.\_denominations, \_counts + other.\_counts);

}

Money Money::operator-(const Money& other) const

{

std::cout << "operator- called" << std::endl;

{

if (\_denominations < 0)

return Money(0, \_counts - other.\_counts);

if (\_counts < 0)

return Money(\_denominations - other.\_denominations, 0);

return Money(\_denominations - other.\_denominations, \_counts - other.\_counts);

}

}

Money& Money::operator=(const Money& other)

{

std::cout << "operator= called" << std::endl;

if (this != &other) {

\_denominations = other.\_denominations;

\_counts = other.\_counts;

}

return \*this;

}

Money& Money::operator=(Money&& other) noexcept

{

std::cout << "operator=(move) called" << std::endl;

if (this != &other) {

\_denominations = std::move(other.\_denominations);

\_counts = std::move(other.\_counts);

other.\_denominations = 0;

other.\_counts = 0;

}

return \*this;

}

bool Money::operator==(const Money& other) const

{

std::cout << "operator== called" << std::endl;

if (\_denominations == other.\_denominations && \_counts == other.\_counts)

return true;

else

return false;

}

bool Money::operator!=(const Money& other) const

{

std::cout << "operator!= called" << std::endl;

if (\_denominations == other.\_denominations || \_counts == other.\_counts)

return false;

else

return true;

}

bool Money::operator<(const Money& other) const

{

std::cout << "operator< called" << std::endl;

if (\_denominations < other.\_denominations && \_counts < other.\_counts)

return true;

else if (\_denominations == other.\_denominations && \_counts < other.\_counts)

return true;

else if (\_denominations < other.\_denominations && \_counts == other.\_counts)

return true;

else return false;

}

bool Money::operator<=(const Money& other) const

{

std::cout << "operator<= called" << std::endl;

if (\_denominations <= other.\_denominations && \_counts <= other.\_counts)

return true;

else return false;

}

bool Money::operator>(const Money& other) const

{

std::cout << "operator> called" << std::endl;

if (\_denominations > other.\_denominations && \_counts > other.\_counts)

return true;

else if (\_denominations == other.\_denominations && \_counts > other.\_counts)

return true;

else if (\_denominations > other.\_denominations && \_counts == other.\_counts)

return true;

else return false;

}

bool Money::operator>=(const Money& other) const

{

std::cout << "operator>= called" << std::endl;

if (\_denominations >= other.\_denominations && \_counts >= other.\_counts)

return true;

else return false;

}

Money::operator std::string() const

{

std::cout << "operator std::string() called" << std::endl;

std::string a = "\_denominations " + std::to\_string(\_denominations);

a += " \_counts " + std::to\_string(\_counts);

return a;

}

Money& Money::operator++() // Prefix increment

{

std::cout << "operator++ called" << std::endl;

++\_counts;

return \*this;

}

Money Money::operator++(int) // Postfix increment

{

std::cout << "operator++(int) called" << std::endl;

Money temp = \*this;

++\_counts;

return temp;

}

Money& Money::operator--() // Prefix decrement

{

std::cout << "operator-- called" << std::endl;

if (\_counts > 0)

--\_counts;

return \*this;

}

Money Money::operator--(int) // Postfix decrement

{

std::cout << "operator--(int) called" << std::endl;

Money temp = \*this;

if (\_counts > 0)

--\_counts;

return temp;

}

std::ostream& operator<<(std::ostream& os, const Money& money) {

std::cout << "operator<<(ostream) called" << std::endl;

return os << "Denomination: " << money.\_denominations << std::endl << "Counts: " << money.\_counts << std::endl;

}

std::istream& operator>>(std::istream& is, Money& money) {

std::cout << "operator>>(istream) called" << std::endl;

return is >> money.\_denominations >> money.\_counts;

}

#pragma endregion

Main.cpp

#include <iostream>

#include "Money.h"

int main()

{

std::cout << "----------------------- Constructors ------------------------" << std::endl;

Money walletOne(10, 5);

Money walletTwo(50, 200);

Money walletThree = walletTwo;

Money walletFour = std::move(walletThree);

std::cout << std::endl;

std::cout << "----------------------- Overloaded operators ------------------------" << std::endl;

Money walletFive(100, 20);

Money walletSix(100, 50);

Money sum = walletSix + walletFive;

std::cout << "walletSix + walletFive:" << std::endl;

std::cout << sum << std::endl;

Money diff = walletSix - walletFive;

std::cout << "walletSix - walletFive:" << std::endl;

std::cout << diff << std::endl;

walletThree = walletTwo;

std::cout << "walletThree after assigmetn walletTwo:" << std::endl;

std::cout << walletThree << std::endl;

std::cout << std::boolalpha;

std::cout << "walletFive == walletSix ? " << (walletFive == walletSix) << std::endl;

std::cout << "walletFive != walletSix ? " << (walletFive != walletSix) << std::endl;

++walletFive;

std::cout << "walletFive after ++ : " << walletFive << std::endl;

walletFive--;

std::cout << "walletFive after -- : " << walletFive << std::endl;

std::string str = static\_cast<std::string>(walletSix);

std::cout << "walletSix as a string: " << str << std::endl;

}

Результат



Завдання 2

Windows Form

**PhoneClass.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab\_WinForm\_Daryev

{

internal class PhoneClass

{

public string phoneName { get; set; }

public string phoneModel { get; set; }

public DateTime phoneYear { get; set; }

public double phonePrice { get; set; }

public PhoneClass()

{

phoneName = "default";

phoneModel = "default";

phoneYear = new DateTime(1960, 01, 01);

phonePrice = 0;

}

public PhoneClass(string name, string number, DateTime year, double price)

{

phoneName = name;

phoneModel = number;

if (year <= new DateTime(1960, 01, 01))

{

phoneYear = new DateTime(1960, 01, 01);

}

else if (year >= DateTime.Now)

{

phoneYear = DateTime.Now;

}

else phoneYear = year;

phonePrice = price;

}

// ----------------------------- Operator Overloading -----------------------------

static public bool operator >(PhoneClass phone1, PhoneClass phone2)

{

return phone1.phonePrice > phone2.phonePrice;

}

static public bool operator <(PhoneClass phone1, PhoneClass phone2)

{

return phone1.phonePrice < phone2.phonePrice;

}

static public bool operator ==(PhoneClass phone1, PhoneClass phone2)

{

return phone1.Equals(phone2);

}

static public bool operator !=(PhoneClass phone1, PhoneClass phone2)

{

return !phone1.Equals(phone2);

}

static public bool operator >=(PhoneClass phone1, PhoneClass phone2)

{

return phone1.phonePrice >= phone2.phonePrice;

}

static public bool operator <=(PhoneClass phone1, PhoneClass phone2)

{

return phone1.phonePrice <= phone2.phonePrice;

}

public override bool Equals(object obj)

{

if (obj == null)

return false;

if (obj is PhoneClass phone)

{

return this.phoneName.Equals(phone.phoneName, StringComparison.OrdinalIgnoreCase)

&& this.phoneModel.Equals(phone.phoneModel, StringComparison.OrdinalIgnoreCase)

&& this.phoneYear == phone.phoneYear

&& this.phonePrice == phone.phonePrice;

}

return false;

}

public override string ToString()

{

return string.Format("{0,-15} {1,-15} {2,10} {3,10}",

this.phoneName, this.phoneModel, this.phoneYear, this.phonePrice);

}

public override int GetHashCode()

{

return HashCode.Combine(phoneName.GetHashCode(), phoneModel.GetHashCode(), phoneYear.GetHashCode(), phonePrice.GetHashCode());

}

static public PhoneClass operator ++(PhoneClass phone)

{

phone.phonePrice++;

return phone;

}

static public PhoneClass operator --(PhoneClass phone)

{

if (phone.phonePrice > 0)

phone.phonePrice--;

return phone;

}

static public PhoneClass operator +(PhoneClass phone, double value)

{

phone.phonePrice += value;

return phone;

}

static public PhoneClass operator -(PhoneClass phone, double value)

{

if (phone.phonePrice - value >= 0)

phone.phonePrice -= value;

else

phone.phonePrice = 0;

return phone;

}

//---------------------------------------------------------------------------------

}

}

MainForm.cs

using System.Data;

using System.IO;

using System.Numerics;

namespace Lab\_WinForm\_Daryev

{

public partial class MainForm : Form

{

private List<PhoneClass> phoneList;

private DataTable DT;

private DataTable filteredDT; // Filter results

private DataTable searchDT; // Search results

private int classCounter;

// -------------------------- Constructor ---------------------------

public MainForm()

{

InitializeComponent();

phoneList = new List<PhoneClass>();

DT = new DataTable();

DT.Columns.Add("Company", typeof(string));

DT.Columns.Add("Phone model", typeof(string));

DT.Columns.Add("Phone year", typeof(DateTime));

DT.Columns.Add("Price", typeof(int));

filteredDT = DT.Clone();

searchDT = DT.Clone();

dataGridView1.DataSource = DT;

}

// ------------------------ Input controls -------------------------

private void ControlInput(object sender, KeyPressEventArgs e)

{

switch (sender)

{

case TextBox tb:

switch (tb.Name)

{

case "modelFilterTB":

case "nameFilterTB":

case "phNameTB":

case "phModelTB":

if (e.KeyChar != '-' && !char.IsDigit(e.KeyChar) &&

!char.IsLetter(e.KeyChar) && !char.IsControl(e.KeyChar))

e.Handled = true;

break;

case "phYearMTB":

case "yearFilterTB":

if (!char.IsDigit(e.KeyChar) && !char.IsControl(e.KeyChar))

e.Handled = true;

break;

case "phPriceTB":

case "priceFilterTB":

if (!char.IsDigit(e.KeyChar) && !char.IsControl(e.KeyChar) && e.KeyChar != '.')

e.Handled = true;

if (e.KeyChar == '.')

{

if (tb.Text.Contains('.') || tb.SelectionStart == 0)

e.Handled = true;

}

break;

}

break;

case MaskedTextBox mtb:

if (!char.IsDigit(e.KeyChar) && !char.IsControl(e.KeyChar))

e.Handled = true;

break;

}

}

private void searchTBControlInput(object sender, KeyPressEventArgs e)

{

if (searchByYearRB.Checked)

{

if (!char.IsDigit(e.KeyChar) && !char.IsControl(e.KeyChar))

e.Handled = true;

}

else

{

if (e.KeyChar != '-' && !char.IsDigit(e.KeyChar) &&

!char.IsLetter(e.KeyChar) && !char.IsControl(e.KeyChar))

e.Handled = true;

}

}

// --------------------- Update class counter -----------------------

private void UpdateCounter()

{

classCountL.Text = phoneList.Count.ToString();

}

// ---------------------------- Events ------------------------------

// Create object

private void createButton\_Click(object sender, EventArgs e)

{

if (withParChB.CheckState == CheckState.Checked)

{

var phOne = new PhoneClass();

phoneList.Add(phOne);

DT.Rows.Add(phOne.phoneName, phOne.phoneModel, phOne.phoneYear, phOne.phonePrice);

UpdateCounter();

return;

}

if (!ValidateInputs(out string company, out string model, out DateTime year, out double price))

return;

var phone = new PhoneClass(company, model, year, price);

phoneList.Add(phone);

DT.Rows.Add(phone.phoneName, phone.phoneModel, phone.phoneYear, phone.phonePrice);

phNameTB.Clear();

phModelTB.Clear();

phYearMTB.Clear();

phPriceTB.Clear();

UpdateCounter();

}

// Save file

private void saveFileTSMI\_Click(object sender, EventArgs e)

{

var saveFile = new SaveFileDialog();

string filePath;

saveFile.Filter = "Binary File(.bin)|\*.bin";

if (saveFile.ShowDialog() == DialogResult.OK)

{

filePath = saveFile.FileName;

using (var binWritter = new BinaryWriter(new FileStream(filePath, FileMode.Create)))

{

foreach (var value in phoneList)

{

binWritter.Write(value.phoneName);

binWritter.Write(value.phoneModel);

binWritter.Write(value.phoneYear.Date.ToBinary());

binWritter.Write(value.phonePrice);

}

}

}

}

// Open file

private void openFileTSMI\_Click(object sender, EventArgs e)

{

var openFile = new OpenFileDialog();

string filePath;

openFile.Filter = "Binary File(.bin)|\*.bin";

if (openFile.ShowDialog() == DialogResult.OK)

{

filePath = openFile.FileName;

phoneList.Clear();

try

{

using (var binReader = new BinaryReader(new FileStream(filePath, FileMode.Open)))

{

while (binReader.BaseStream.Position < binReader.BaseStream.Length)

{

var value = new PhoneClass

{

phoneName = binReader.ReadString(),

phoneModel = binReader.ReadString(),

phoneYear = DateTime.FromBinary(binReader.ReadInt64()),

phonePrice = binReader.ReadDouble()

};

phoneList.Add(value);

}

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

foreach (var phone in phoneList)

{

DT.Rows.Add(phone.phoneName, phone.phoneModel, phone.phoneYear, phone.phonePrice);

}

}

}

private void filterButton\_Click(object sender, EventArgs e)

{

var filteredList = new List<PhoneClass>();

foreach (var phone in phoneList)

{

bool matches = true;

if (nameFilterCB.Checked && !string.IsNullOrWhiteSpace(nameFilterTB.Text))

{

if (!phone.phoneName.Equals(nameFilterTB.Text, StringComparison.OrdinalIgnoreCase))

matches = false;

}

if (modelFilterCB.Checked && !string.IsNullOrWhiteSpace(modelFilterTB.Text))

{

if (!phone.phoneModel.Equals(modelFilterTB.Text, StringComparison.OrdinalIgnoreCase))

matches = false;

}

if (yearFilterCB.Checked && DateTime.TryParse(yearFilterTB.Text, out DateTime year))

{

if (phone.phoneYear != year)

matches = false;

}

if (priceFilterCB.Checked && double.TryParse(priceFilterTB.Text, out double price))

{

if (phone.phonePrice != price)

matches = false;

}

if (matches)

filteredList.Add(phone);

}

filteredDT.Clear();

foreach (var phone in filteredList)

{

filteredDT.Rows.Add(phone.phoneName, phone.phoneModel, phone.phoneYear, phone.phonePrice);

}

dataGridView1.DataSource = filteredDT;

}

private void resetButtonTSMI\_Click(object sender, EventArgs e)

{

dataGridView1.DataSource = DT;

}

private void searchButton\_Click(object sender, EventArgs e)

{

var searchList = new List<PhoneClass>();

DateTime year;

double price;

foreach (var phone in phoneList)

{

bool matches = true;

if (searchNameRB.Checked)

{

if (!phone.phoneName.Equals(allSearchValueTB.Text, StringComparison.OrdinalIgnoreCase))

matches = false;

}

if (searchByModelRB.Checked)

{

if (!phone.phoneModel.Equals(allSearchValueTB.Text, StringComparison.OrdinalIgnoreCase))

matches = false;

}

if (searchByYearRB.Checked)

{

if (DateTime.TryParse(yearSearchValTB.Text, out year))

{

if (phone.phoneYear != year)

matches = false;

}

}

if (searchByPriceRB.Checked)

{

if (double.TryParse(allSearchValueTB.Text, out price))

{

if (phone.phonePrice != price)

matches = false;

}

}

if (matches)

searchList.Add(phone);

}

searchDT.Clear();

foreach (var phone in searchList)

{

searchDT.Rows.Add(phone.phoneName, phone.phoneModel, phone.phoneYear, phone.phonePrice);

}

dataGridView1.DataSource = searchDT;

}

private void searchByYearRB\_CheckedChanged(object sender, EventArgs e)

{

if (searchByYearRB.Checked)

{

yearSearchValTB.Visible = true;

allSearchValueTB.Visible = false;

}

else

{

yearSearchValTB.Visible = false;

allSearchValueTB.Visible = true;

}

}

// --------------------------- Methodes ----------------------------

// Validate parametrs

private bool ValidateInputs(out string company, out string model, out DateTime year, out double price)

{

company = phNameTB.Text.Trim();

model = phModelTB.Text.Trim();

year = DateTime.MinValue;

price = 0;

if (string.IsNullOrWhiteSpace(company) ||

string.IsNullOrWhiteSpace(model) ||

string.IsNullOrWhiteSpace(phYearMTB.Text) ||

string.IsNullOrWhiteSpace(phPriceTB.Text))

{

MessageBox.Show("Please, fill all fields", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

return false;

}

if (!DateTime.TryParse(phYearMTB.Text, out year))

{

MessageBox.Show("Year is invalid", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

return false;

}

if (year <= new DateTime(1960, 01, 01))

{

year = new DateTime(1960, 01, 01);

}

else if (year >= DateTime.Now)

{

year = DateTime.Now.Date;

}

if (!double.TryParse(phPriceTB.Text, out price) || price <= 0)

{

MessageBox.Show("Price must be a positive number", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

return false;

}

return true;

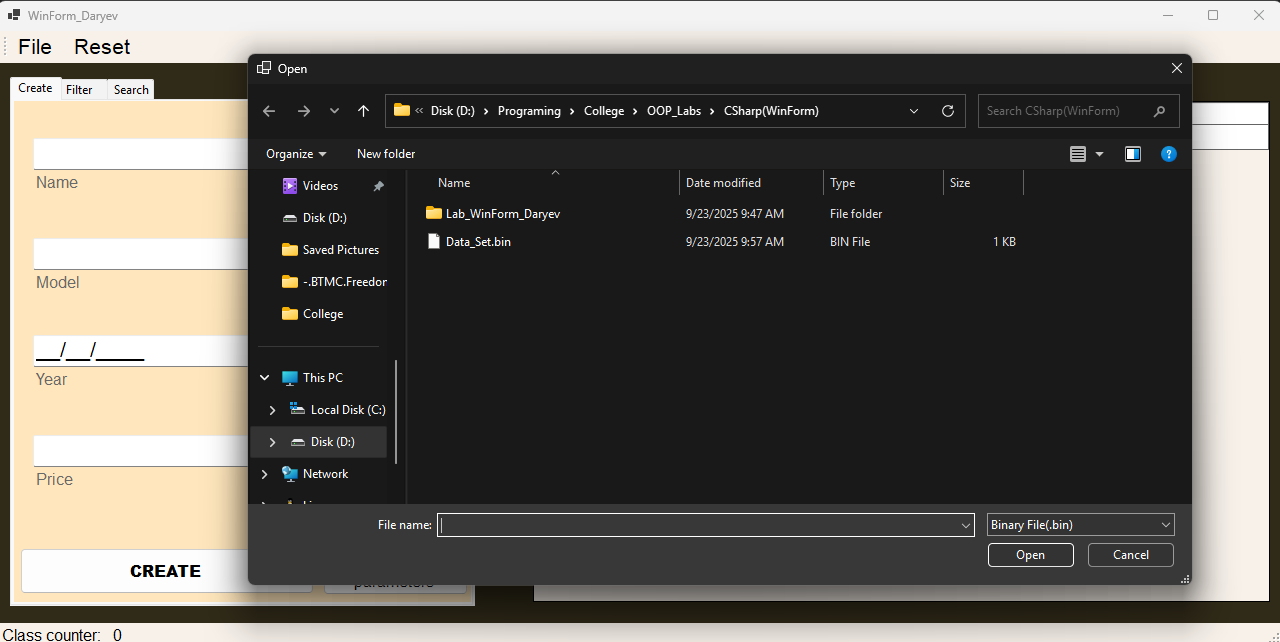
}

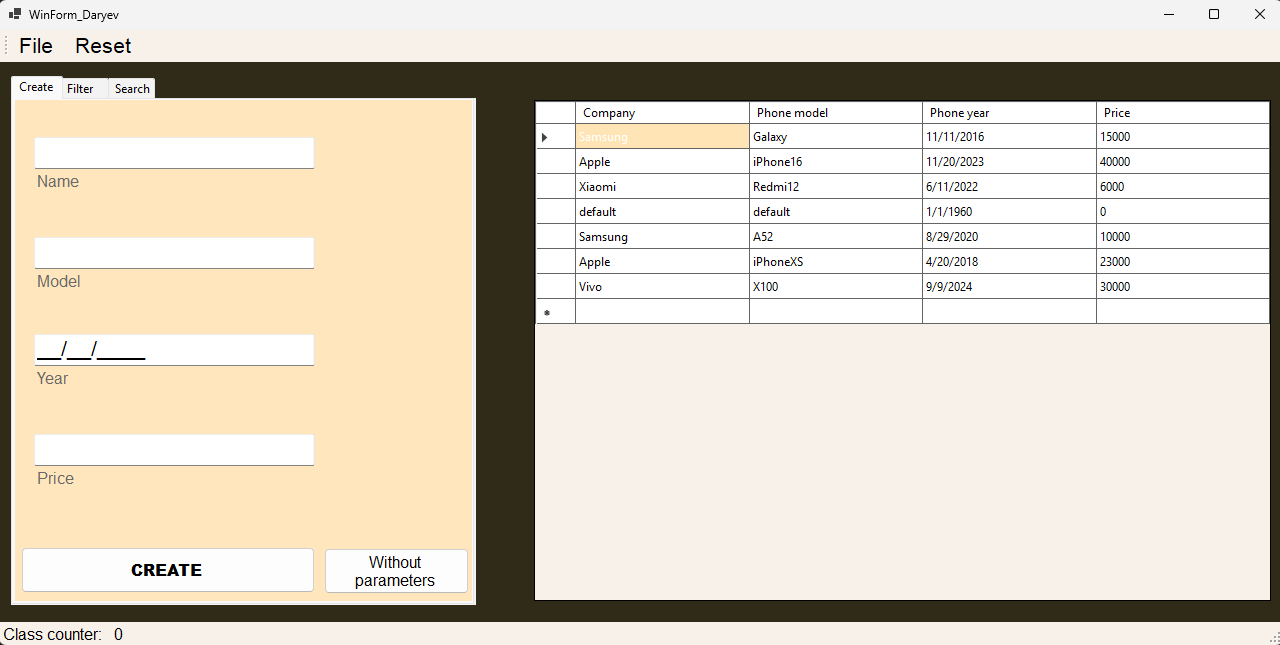
}

}

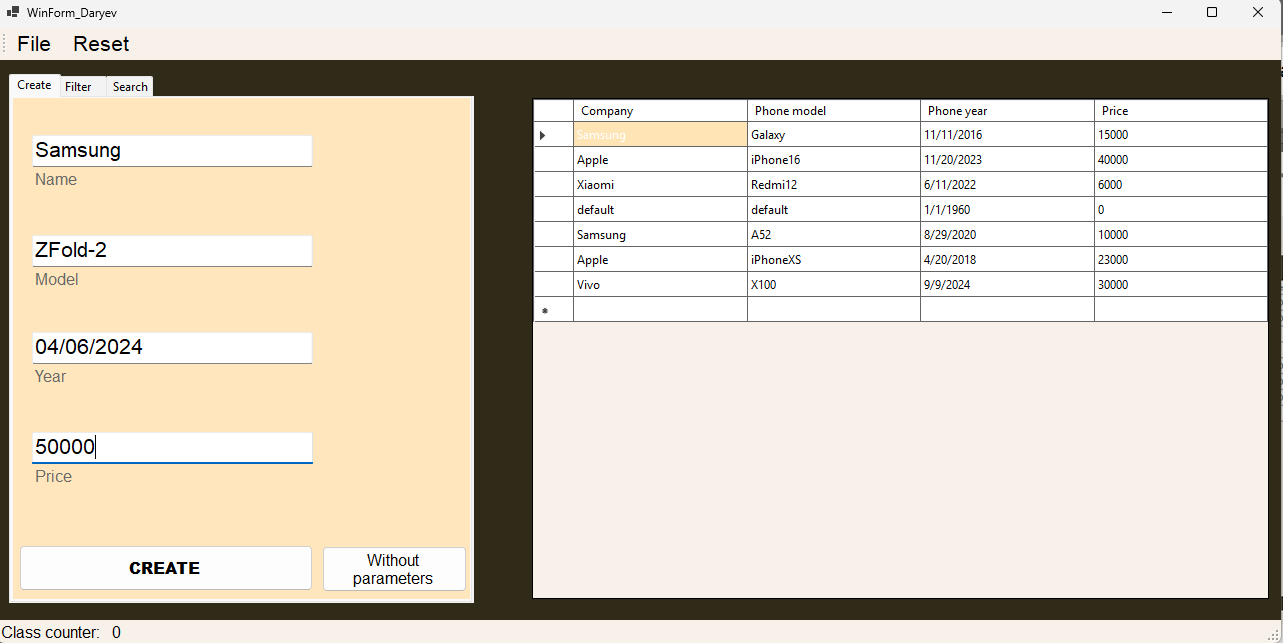
**Результат**

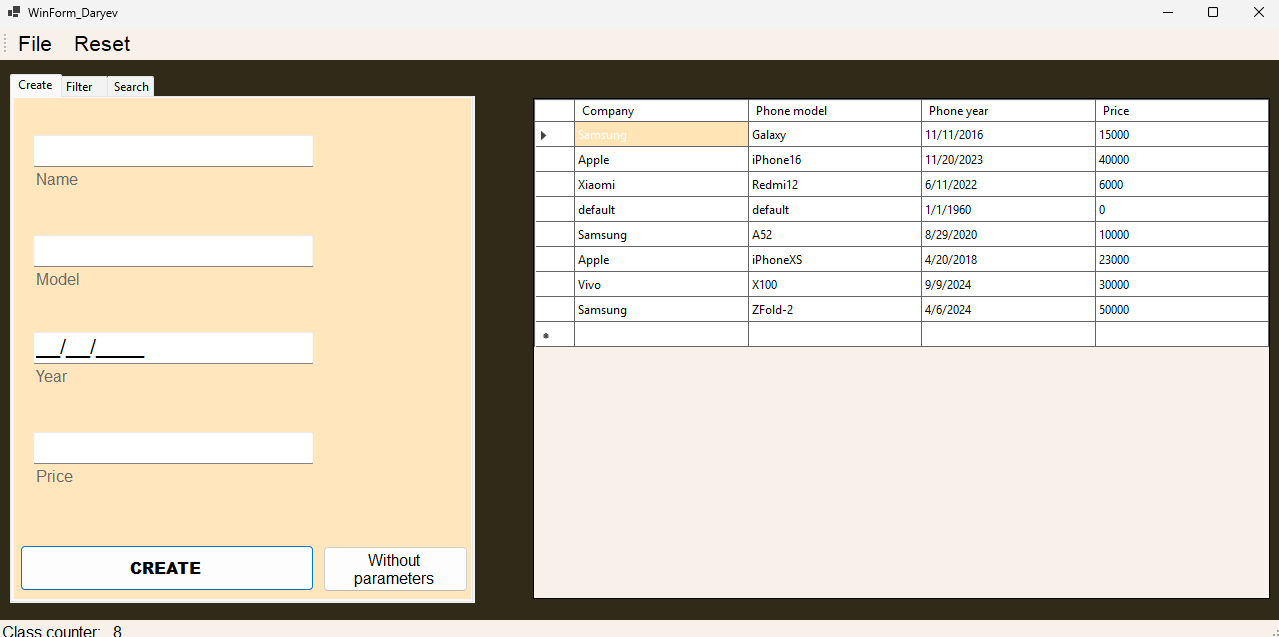
* **Відкриття файлу**

****

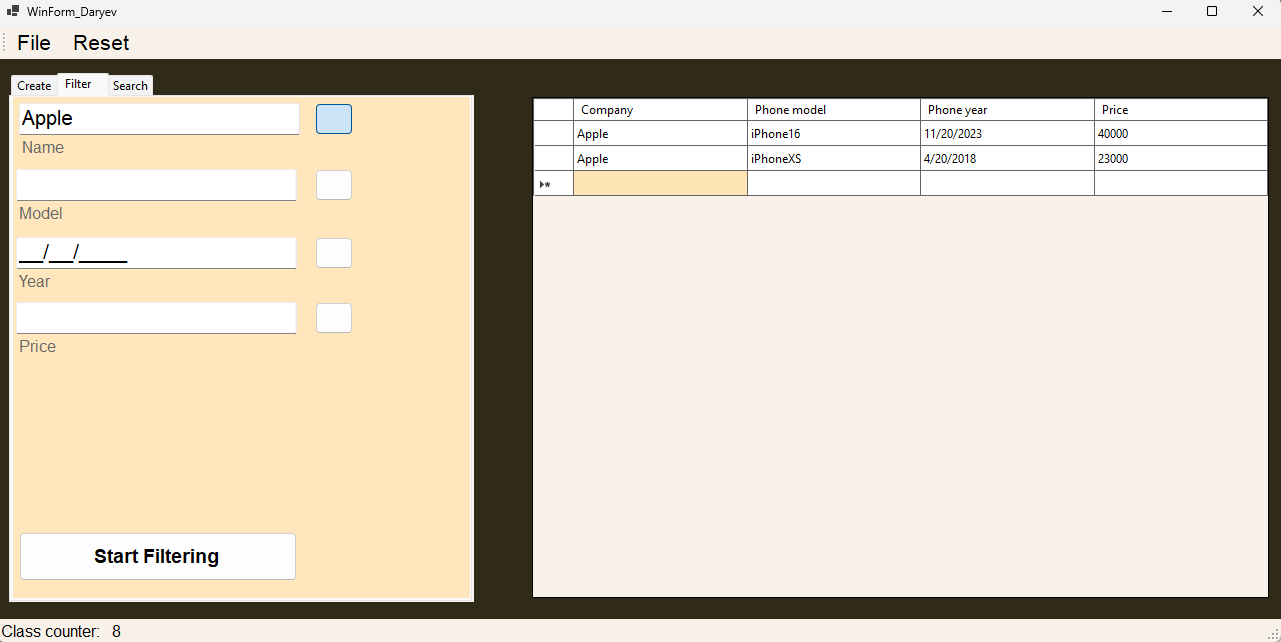
****

* **Додавання нового об’єкта класу**

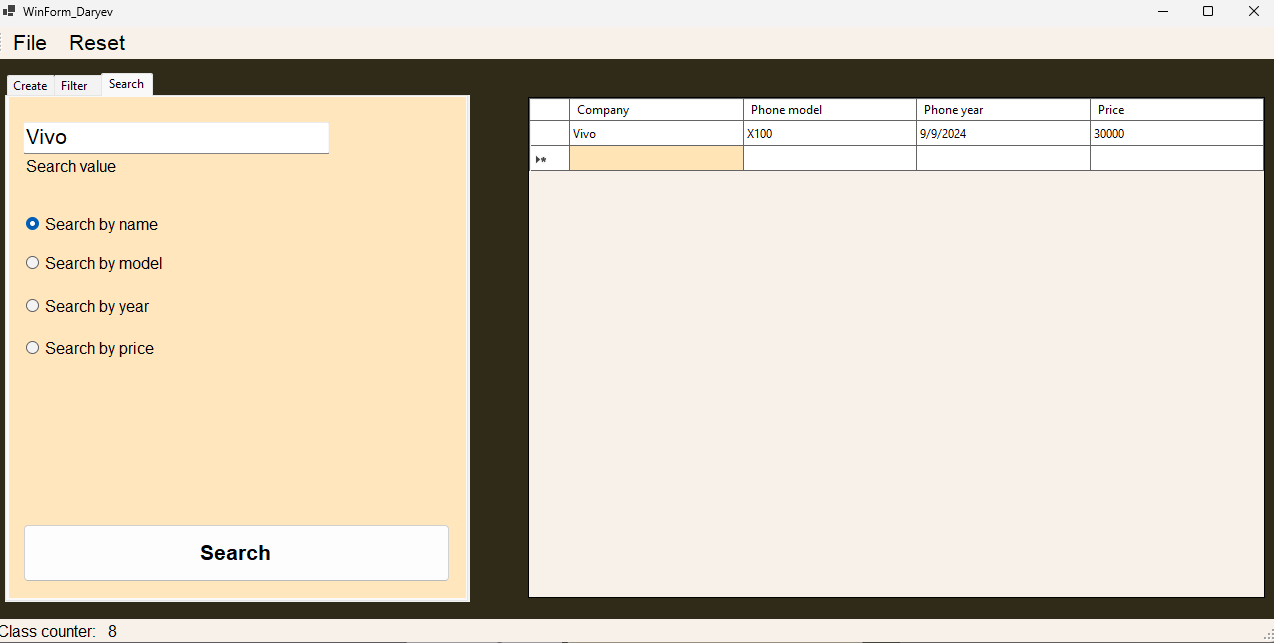
****

****

* **Фільтрація**

****

* **Пошук**

****

Висновок: на лабораторній роботі було одержано практичні навички створення абстрактних типів даних і перевантаження операцій у мові С++ та С#.