МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Навчально-науковий інститут комп’ютерних наук та інформаційних технологій

Кафедра комп’ютерної математики і аналізу даних

Спеціальність комп’ютерні науки

Освітня програма комп’ютерні науки

**Індивідуальне розрахункове завдання**

**пояснювальна записка**

з дисципліни «Кросплатформене програмування»

Виконав студент 2 курсу

групи КН-322б Дядина В. О.

Керівник Марченко І. І.

Оцінка

(національна) (бал) (ECTS)

Харків 2024

Національний технічний університет

«Харківський політехнічний інститут»

Кафедра інформатики та інтелектуальної власності

Дисципліна: кросплатформене програмування

Курс 2, Семестр 4, Група КН-322б

**Завдання**

На індивідуальне розрахункове завдання студенту

Дядині Валентину Олексійовичу

* Створити інтерфейс IName, у якому описується вимога отримання рядкового поля Name. Цей інтерфейс повинен успадковуватися від IComparable.
* Створити ієрархію класів для певної групи товарів магазину:
  + Створити один базовий клас та кілька похідних.
  + Класи мають відрізнятися набором даних.
  + Реалізувати за замовчуванням конструктор і кілька конструкторів з різним набором параметрів.
  + У класах має бути перевантажений метод ToString(), який перетворює на рядок інформацію про об'єкт.
  + Клас має реалізовувати інтерфейси IName.
* Створити клас-дженерик Container для зберігання інформації про об'єкти. У ньому реалізувати функціонал:  
  + додавання об'єкта в контейнер,
  + видалення об'єкта з контейнера за індексом,
  + упорядкування інформації у контейнері,
  + перетворення вмісту контейнера до рядка,
  + отримання об'єкта за індексом (№) та за ім'ям,
  + щоб він був ітерованим по всьому вмісту, а також містив ітератор, який би повертав об'єкти, імена яких починаються на вказаний рядок,
  + виведення вмісту в ньому в текстовий файл на ім'я контейнера.
* Передбачити коректне опрацювання можливих виняткових ситуацій.
* Реалізувати можливість серіалізації/десералізації контейнера.
* У контейнер додати метод сортування, як параметр передавати метод порівняння двох об'єктів; знаходження всіх об'єктів, які задовольняють умову, задану лямбда-виразом.
* Реалізувати можливість отримання сумарної вартості товарів на складі зараз без перегляду вартості товарів (але товар може змінити свою вартість, що призведе до зміни сумарної вартості складу).
* Реалізувати для контейнера такі можливості за допомогою LINQ:
  + знаходження найдешевшого і найдорожчого товару,
  + знаходження середньої вартості товарів для кожної категорії.

**Зміст**

[1. Вступ 6](#_Toc168391472)

[2. Інтерфейс IName 8](#_Toc168391474)

[3. Створення ієрархії класів 9](#_Toc168391475)

[4. Клас-дженерик Container 13](#_Toc168391476)

[5. Опрацювання виняткових ситуацій 18](#_Toc168391477)

[6. Серіалізація та десеріалізація контейнера 23](#_Toc168391478)

[7. Сортування за параметром, пошук за лямбдою 26](#_Toc168391479)

[8. Отримання сумарної вартості 29](#_Toc168391480)

[9. LINGQ 30](#_Toc168391481)

# Вступ

Написання програми для індивідуального розрахункового завдання відбувалось у середовищі програмування Microsoft Visual Studio (рис. 1)

# 

Рисунок 1. Середовище програмування Visual Studio

Для зручності роботи з кодом основна програма була розбита на декілька файлів (рис.2):

* Program.cs (файл основної програми, який компілюється)
* Interfaces.cs (файл, у якому містяться інтерфейси IName, IPrice)
* Product.cs (файл, у якому міститься клас Product, його поля і методи)
* Container.cs (файл, який налічує контейнер Container, його поля і методи)

Тому для коректної роботи цих файлів програми між собою, використовувався простір імен IDZ

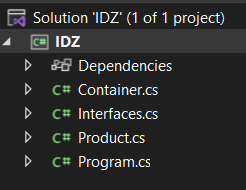


Рисунок 1. Файли програми

# Інтерфейс IName

Інтерфейс — це абстрактний тип, який використовується для визначення набору методів і властивостей, які клас або структура повинні реалізувати. Інтерфейси не можуть містити реалізацію методів, тільки їх підписи (тобто, заголовки методів без тіла).

У коді інтерфейс був створений:

public interface IName: IComparable<IName>

{

string Name { get; }

}

У даному інтерфейсі описується вимога отримання рядкового поля Name. Також він спадковує інтерфейс IComparable<IName>, тобто будь-який клас, який реалізує IName, повинен реалізувати метод CompareTo з інтерфейсу IComparable<IName>:

public int CompareTo(IName? other)

{

if (other == null) return 1;

return this.Name.CompareTo(other.Name);

}

Тобто якщо параметр other не є null, метод порівнює властивість Name поточного об'єкта з властивістю Name об'єкта other за допомогою методу CompareTo класу string. Метод string.CompareTo повертає: Менше нуля, якщо поточний об'єкт менше, ніж інший об'єкт. Нуль, якщо обидва об'єкти рівні. Більше нуля, якщо поточний об'єкт більший, ніж інший об'єкт.

# Створення ієрархії класів

У програмі був створений базовий клас Product з похідними від нього класами Electronics та Clothes.

public class Product : IName, IPrice

{

public string Name { get; private set; }

public decimal Price { get; private set; }

public string Manufacturer { get; private set; }

public Product()

{

Name = "";

Price = 0.0m;

Manufacturer = "";

}

public Product(string name, decimal price)

{

Name = name;

Price = price;

Manufacturer = "";

}

public Product(string name, decimal price, string manufacturer)

{

Name = name;

Price = price;

Manufacturer = manufacturer;

}

public int CompareTo(IName? other)

{

if (other == null) return 1;

return this.Name.CompareTo(other.Name);

}

public override string ToString()

{

return $"Продукт: {Name}, Ціна: {Price:C}, Виробник: {Manufacturer}";

}

}

public class Electronics : Product

{

public int WarrantyPeriod { get; private set; }

public Electronics() : base()

{

WarrantyPeriod = 0;

}

public Electronics(string name, decimal price, int warrantyPeriod)

: base(name, price)

{

WarrantyPeriod = warrantyPeriod;

}

public Electronics(string name, decimal price, string manufacturer, int warrantyPeriod)

: base(name, price, manufacturer)

{

WarrantyPeriod = warrantyPeriod;

}

public override string ToString()

{

return $"{base.ToString()}, Гарантійний термін: {WarrantyPeriod} місяців";

}

}

public class Clothing : Product

{

public string Size { get; private set; }

public string Material { get; private set; }

public Clothing() : base()

{

Size = "";

Material = "";

}

public Clothing(string name, decimal price, string size, string material)

: base(name, price)

{

Size = size;

Material = material;

}

public Clothing(string name, decimal price, string manufacturer, string size, string material)

: base(name, price, manufacturer)

{

Size = size;

Material = material;

}

public override string ToString()

{

return $"{base.ToString()}, Розмір: {Size}, Матеріал: {Material}";

}

}

**Конструктори класу:**

Конструктор за замовчуванням з початковими значеннями для властивостей:   
public Product()

{

Name = "";

Price = 0.0m;

Manufacturer = "";

}

Конструктор з двома параметрами, у якому поле виробник ініціалізується порожнім

public Product(string name, decimal price)

{

Name = name;

Price = price;

Manufacturer = "";

}

Конструктор з трьома параметрами:

public Product(string name, decimal price, string manufacturer)

{

Name = name;

Price = price;

Manufacturer = manufacturer;

}

**Перевантажені методи ToString():**

Метод у базовому класі Product, що повертає назву та ціну:

public override string ToString()

{

return $"Продукт: {Name}, Ціна: {Price:C}, Виробник: {Manufacturer}";

}

Метод у похідному класі Electronics, що повертає гарантійний термін:

public override string ToString()

{

return $"{base.ToString()}, Гарантійний термін: {WarrantyPeriod} місяців";

}

Метод у похідному класі Clothing, що повертає розмір та матеріал:

public override string ToString()

{

return $"{base.ToString()}, Розмір: {Size}, Матеріал: {Material}";

}

**Реалізація інтерфейсу IName:**

public string Name { get; private set; }

Також був доданий інтерфейс IPrice, що використовується для забезпечення узгодженості та стандартизації доступу до властивості Price у різних класах, які можуть мати цю властивість:

public interface IPrice

{

decimal Price { get; }

}

public decimal Price { get; private set; }

# Клас-дженерик Container

У створеному класі-дженерику (клас, що дозволяє створювати класи, які працюють з будь-яким типом даних, забезпечуючи тим самим можливість багаторазового використання коду з різними типами без написання спеціалізованих реалізацій для кожного типу) був доданий наступний функціонал.

**Додавання об’єкта в контейнер:**

public void Add(T item)

{

try

{

if (item == null)

throw new ArgumentNullException(nameof(item), "Об'єкт не може бути null");

T[] newArray = new T[items.Count + 1];

for (int i = 0; i < items.Count; i++)

{

newArray[i] = items[i];

}

newArray[newArray.Length - 1] = item;

items = new List<T>(newArray);

}

catch (ArgumentNullException ex)

{

Console.WriteLine($"Помилка: {ex.Message}");

throw;

}

} }

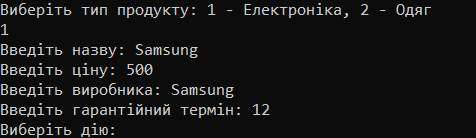


Рисунок 2. Додаввання об’єкта в контейнер

**Видалення за індексом**

public void RemoveAt(int index)

{

try

{

if (index < 0 || index >= items.Count)

throw new IndexOutOfRangeException("Недійсний індекс!");

T[] newArray = new T[items.Count - 1];

for (int i = 0, j = 0; i < items.Count; i++)

{

if (i != index)

{

newArray[j++] = items[i];

}

}

items = new List<T>(newArray);

}

catch (IndexOutOfRangeException ex)

{

Console.WriteLine($"Помилка: {ex.Message}");

throw;

}

}

Рисунок 3. Видалення об’єкта з контейнера

**Упорядкування контейнера**

public void Sort(Comparison<T> comparison)

{

try

{

for (int i = 0; i < items.Count - 1; i++)

{

for (int j = 0; j < items.Count - i - 1; j++)

{

if (comparison(items[j], items[j + 1]) > 0)

{

T temp = items[j];

items[j] = items[j + 1];

items[j + 1] = temp;

}

}

}

}

catch (ArgumentException ex)

{

throw new ArgumentException("Сортування невдале!", ex);

}

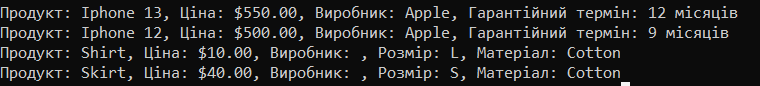
}

Рисунок 4. Об’єкти до упорядкування

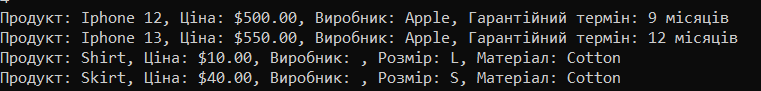


Рисунок 5. Об’єкти після упорядкування

Метод Sort використовує реалізацію IComparable для порівняння елементів та впорядкування їх у порядку зростання

**Перетворення вмісту контейнера до рядка**

public override string ToString()

{

try

{

return string.Join(Environment.NewLine, items);

}

catch (Exception ex)

{

throw new Exception("Не вдалося перетворити контейнер на рядок!", ex);

}

}

**Отримання об’єкта за індексом та за ім’ям**

public T this[int index]

{

get

{

try

{

return items[index];

}

catch (ArgumentOutOfRangeException)

{

throw new IndexOutOfRangeException("Недійсний індекс!");

}

}

}

public T? GetByName(string name)

{

try

{

return items.FirstOrDefault(item => item.Name == name);

}

catch (Exception ex)

{

throw new Exception("Помилка отримання елемента за назвою", ex);

}

}



Рисунок 6. Пошук за ім’ям

**Пошук за префіксом**

public IEnumerator<T> GetEnumerator()

{

return items.GetEnumerator();

}

IEnumerator IEnumerable.GetEnumerator()

{

return GetEnumerator();

}

public IEnumerable<T> GetItemsByPrefix(string prefix)

{

if (string.IsNullOrEmpty(prefix))

throw new ArgumentException("Префікс не може бути нульовим або порожнім", nameof(prefix));

return GetItemsByPrefixInternal(prefix);

}

private IEnumerable<T> GetItemsByPrefixInternal(string prefix)

{

foreach (var item in items)

{

if (item.Name.StartsWith(prefix, StringComparison.OrdinalIgnoreCase))

{

yield return item;

}

}

}

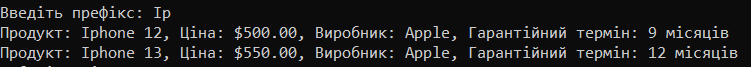


Рисунок 6. Пошук за префіксом

GetEnumerator() метод повертає ітератор для елементів у контейнері. GetItemsByPrefix(string prefix) метод повертає всі елементи в контейнері, чиї імена починаються з певного префіксу. GetItemsByPrefixInternal метод реалізований як приватний і використовує yield return для повернення елементів, що відповідають критерію. Він ітерує через всі елементи у списку items і перевіряє, чи починається Name кожного елемента з вказаного префіксу.

**Виведення вмісту в ньому в текстовий файл на ім'я контейнера.**

public void SaveToFile()

{

try

{

File.WriteAllText($"{containerName}.txt", ToString());

Console.WriteLine("Вміст контейнера збережено до файлу Products.txt");

}

catch (IOException ex)

{

throw new IOException("Не вдалося зберегти контейнер у файл", ex);

}

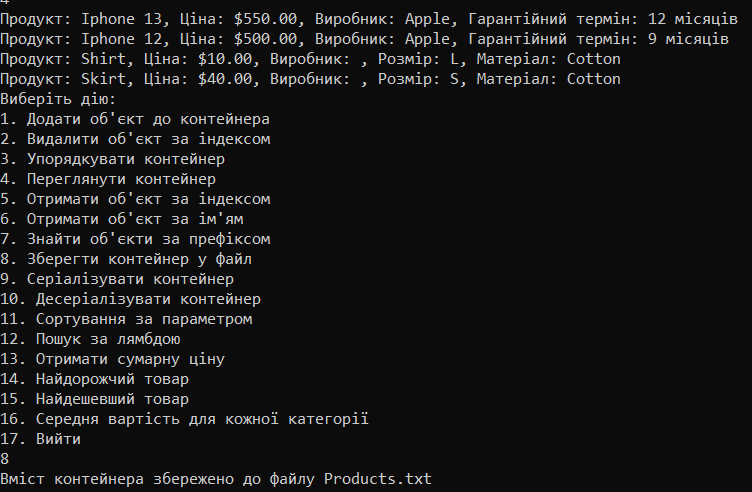
catch (Exception ex)

{

throw new Exception("Неочікувана помилка під час збереження у файл", ex);

}

}



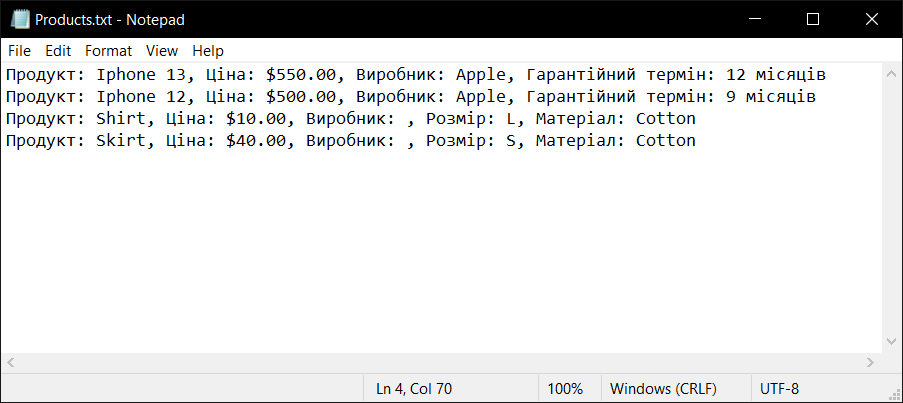


Рисунок 7. Робота з текстовим файлом

# Опрацювання виняткових ситуацій

public void Add(T item)

{

try

{

if (item == null)

throw new ArgumentNullException(nameof(item), "Об'єкт не може бути null");

T[] newArray = new T[items.Count + 1];

for (int i = 0; i < items.Count; i++)

{

newArray[i] = items[i];

}

newArray[newArray.Length - 1] = item;

items = new List<T>(newArray);

}

catch (ArgumentNullException ex)

{

Console.WriteLine($"Помилка: {ex.Message}");

throw;

}

}

Викидається, якщо метод намагається додати null елемент до контейнера, що заборонено.

public void RemoveAt(int index)

{

try

{

if (index < 0 || index >= items.Count)

throw new IndexOutOfRangeException("Недійсний індекс!");

T[] newArray = new T[items.Count - 1];

for (int i = 0, j = 0; i < items.Count; i++)

{

if (i != index)

{

newArray[j++] = items[i];

}

}

items = new List<T>(newArray);

}

catch (IndexOutOfRangeException ex)

{

Console.WriteLine($"Помилка: {ex.Message}");

throw;

}

}Викидається, якщо метод намагається видалити елемент за недійсним індексом.

public void Sort()

{

try

{

items.Sort();

}

catch (InvalidOperationException ex)

{

throw new InvalidOperationException("Сортування невдале!", ex);}

Викидається, якщо сортування невдале через те, що об'єкти у списку не підтримують порівняння.

public void Sort(Comparison<T> comparison)

{

try

{

items.Sort(comparison);

}

catch (ArgumentException ex)

{

throw new ArgumentException("Сортування невдале!", ex);

}

}

Викидається, якщо сортування невдале через помилку в порівнянні об'єктів.

public void Sort()

{

public override string ToString()

{

try

{

return string.Join(Environment.NewLine, items);

}

catch (Exception ex)

{

throw new Exception("Не вдалося перетворити контейнер на рядок!", ex);

}

}

Обгортає будь-яку можливу помилку при конвертації контейнера в рядок у загальний виняток з зрозумілим повідомленням.

public T this[int index]

{

get

{

try

{

return items[index];

}

catch (ArgumentOutOfRangeException)

{

throw new IndexOutOfRangeException("Недійсний індекс!");

}

}

}

Викидається, якщо звертаються до недійсного індексу. Обгорнутий у IndexOutOfRangeException з зрозумілим повідомленням.

public T? GetByName(string name)

{

try

{

return items.FirstOrDefault(item => item.Name == name);

}

catch (Exception ex)

{

throw new Exception("Помилка отримання елемента за назвою", ex);

}

}

Обгортає будь-яку можливу помилку при пошуку елемента за назвою у загальний виняток з зрозумілим повідомленням.

private IEnumerable<T> GetItemsByPrefixInternal(string prefix)

{

foreach (var item in items)

{

if (item.Name.StartsWith(prefix, StringComparison.OrdinalIgnoreCase))

{

yield return item;

}

}

}

Викидається, якщо переданий порожній або нульовий префікс.

public void SaveToFile()

{

try

{

File.WriteAllText($"{containerName}.txt", ToString());

Console.WriteLine("Вміст контейнера збережено до файлу Products.txt");

}

catch (IOException ex)

{

throw new IOException("Не вдалося зберегти контейнер у файл", ex);

}

catch (Exception ex)

{

throw new Exception("Неочікувана помилка під час збереження у файл", ex);

}

Обробляє помилки при записі файлу. IOException для помилок вводу/виводу, Exception для загальних помилок.

public void Serialize(string filePath)

{

try

{

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

using (var writer = new BinaryWriter(fs))

{

writer.Write(containerName);

writer.Write(items.Count);

foreach (var item in items)

{

if (item is Product product)

{

writer.Write(product.GetType().FullName ?? string.Empty);

writer.Write(product.Name);

writer.Write(product.Price);

writer.Write(product.Manufacturer);

if (product is Electronics electronics)

{

writer.Write(electronics.WarrantyPeriod);

}

else if (product is Clothing clothing)

{

writer.Write(clothing.Size);

writer.Write(clothing.Material);

}

}

}

}

}

catch (IOException ex)

{

throw new IOException("Не вдалося серіалізувати контейнер", ex);

}

catch (Exception ex)

{

throw new Exception("Неочікувана помилка під час серіалізації", ex);

}

}

Обробляє помилки при серіалізації контейнера. IOException для помилок вводу/виводу, Exception для загальних помилок.

public void Deserialize(string filePath)

{

try

{

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

using (var reader = new BinaryReader(fs))

{

containerName = reader.ReadString();

int count = reader.ReadInt32();

items = new List<T>(count);

for (int i = 0; i < count; i++)

{

string typeName = reader.ReadString();

string name = reader.ReadString();

decimal price = reader.ReadDecimal();

string manufacturer = reader.ReadString();

if (typeName == typeof(Electronics).FullName)

{

int warrantyPeriod = reader.ReadInt32();

items.Add((T)(IName)new Electronics(name, price, manufacturer, warrantyPeriod));

}

else if (typeName == typeof(Clothing).FullName)

{

string size = reader.ReadString();

string material = reader.ReadString();

items.Add((T)(IName)new Clothing(name, price, manufacturer, size, material));

}

}

}

}

catch (IOException ex)

{

throw new IOException("Не вдалося десеріалізувати контейнер", ex);

}

catch (Exception ex)

{

throw new Exception("Неочікувана помилка під час десеріалізації", ex);

}

}

Обробляє помилки при десеріалізації контейнера. IOException для помилок вводу/виводу, Exception для загальних помилок.

public T FindCheapestItem()

{

if (!items.Any())

throw new InvalidOperationException("Контейнер порожній");

return items.OrderBy(item => item.Price).First();

}

public T FindMostExpensiveItem()

{

if (!items.Any())

throw new InvalidOperationException("Контейнер порожній");

return items.OrderByDescending(item => item.Price).First();

}

Викидається, якщо метод намагається знайти найдешевший або найдорожчий елемент у порожньому контейнері.

# Серіалізація та десеріалізація контейнера

Серіалізація - це процес перетворення об'єктів у потік байтів для їх збереження або передачі, а десеріалізація - це оборотне перетворення потоку байтів назад у об'єкти. Методи контейнеру, що відповідають ці процеси у коді зображені нище. Длясеріализації та десеріалізації використовуть Binary Writer, Binary Reader:

public void Serialize(string filePath)

{

try

{

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

using (var writer = new BinaryWriter(fs))

{

writer.Write(containerName);

writer.Write(items.Count);

foreach (var item in items)

{

if (item is Product product)

{

writer.Write(product.GetType().FullName ?? string.Empty);

writer.Write(product.Name);

writer.Write(product.Price);

writer.Write(product.Manufacturer);

if (product is Electronics electronics)

{

writer.Write(electronics.WarrantyPeriod);

}

else if (product is Clothing clothing)

{

writer.Write(clothing.Size);

writer.Write(clothing.Material);

}

}

}

}

}

catch (IOException ex)

{

throw new IOException("Не вдалося серіалізувати контейнер", ex);

}

catch (Exception ex)

{

throw new Exception("Неочікувана помилка під час серіалізації", ex);

}

}

public void Deserialize(string filePath)

{

try

{

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

using (var reader = new BinaryReader(fs))

{

containerName = reader.ReadString();

int count = reader.ReadInt32();

items = new List<T>(count);

for (int i = 0; i < count; i++)

{

string typeName = reader.ReadString();

string name = reader.ReadString();

decimal price = reader.ReadDecimal();

string manufacturer = reader.ReadString();

if (typeName == typeof(Electronics).FullName)

{

int warrantyPeriod = reader.ReadInt32();

items.Add((T)(IName)new Electronics(name, price, manufacturer, warrantyPeriod));

}

else if (typeName == typeof(Clothing).FullName)

{

string size = reader.ReadString();

string material = reader.ReadString();

items.Add((T)(IName)new Clothing(name, price, manufacturer, size, material));

}

}

}

}

catch (IOException ex)

{

throw new IOException("Не вдалося десеріалізувати контейнер", ex);

}

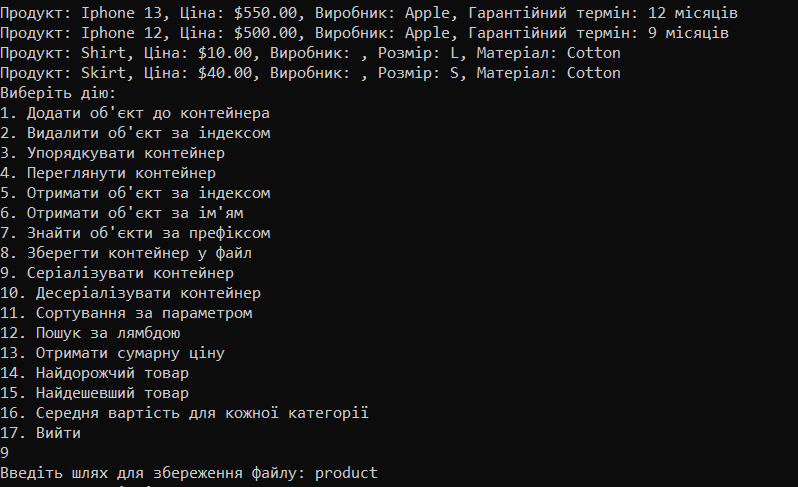
catch (Exception ex)

{

throw new Exception("Неочікувана помилка під час десеріалізації", ex);

}

}



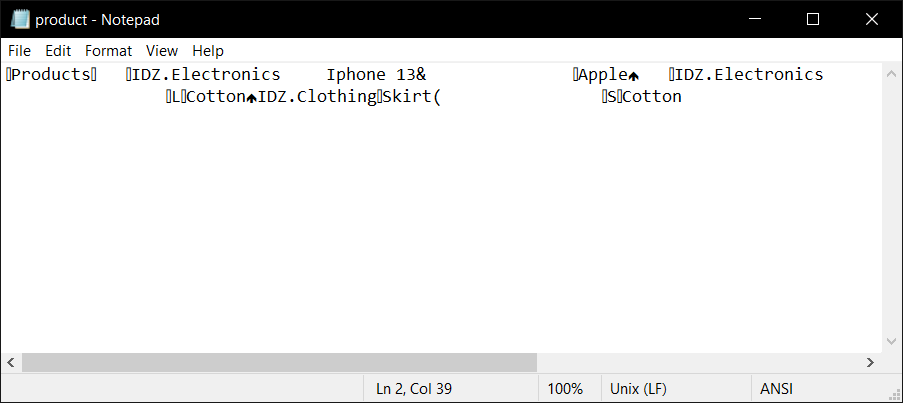


Рисунок 8. Серіалізація бінарного файлу

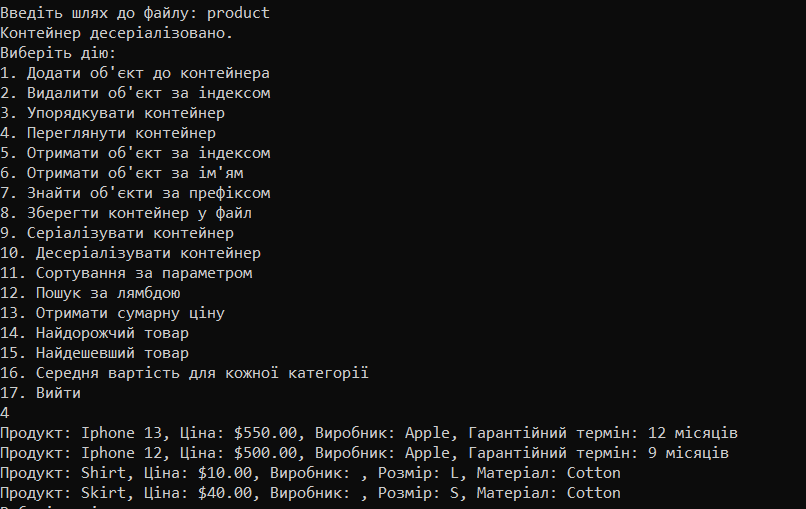


Рисунок 9. Десеріалізація бінарного файлу

# Сортування за параметром, пошук за лямбдою

Функція сортування за параметром у головній програмі:

static void SortItems(Container<Product> container)

{

Console.WriteLine("Виберіть спосіб сортування:");

Console.WriteLine("1. За ціною (зростання)");

Console.WriteLine("2. За ціною (спадання)");

Console.WriteLine("3. За назвою (зростання)");

Console.WriteLine("4. За назвою (спадання)");

string? choice = Console.ReadLine();

switch (choice)

{

case "1":

container.Sort((x, y) => x.Price.CompareTo(y.Price));

break;

case "2":

container.Sort((x, y) => y.Price.CompareTo(x.Price));

break;

case "3":

container.Sort((x, y) => string.Compare(x.Name, y.Name, StringComparison.Ordinal));

break;

case "4":

container.Sort((x, y) => string.Compare(y.Name, x.Name, StringComparison.Ordinal));

break;

default:

Console.WriteLine("Невірний вибір, сортування не виконано.");

return;

}

Console.WriteLine("Контейнер упорядковано.");

}

Метод сортування за параметром у контейнері:

public void Sort(Comparison<T> comparison)

{

try

{

for (int i = 0; i < items.Count - 1; i++)

{

for (int j = 0; j < items.Count - i - 1; j++)

{

if (comparison(items[j], items[j + 1]) > 0)

{

T temp = items[j];

items[j] = items[j + 1];

items[j + 1] = temp;

}

}

}

}

catch (ArgumentException ex)

{

throw new ArgumentException("Сортування невдале!", ex);

}

}

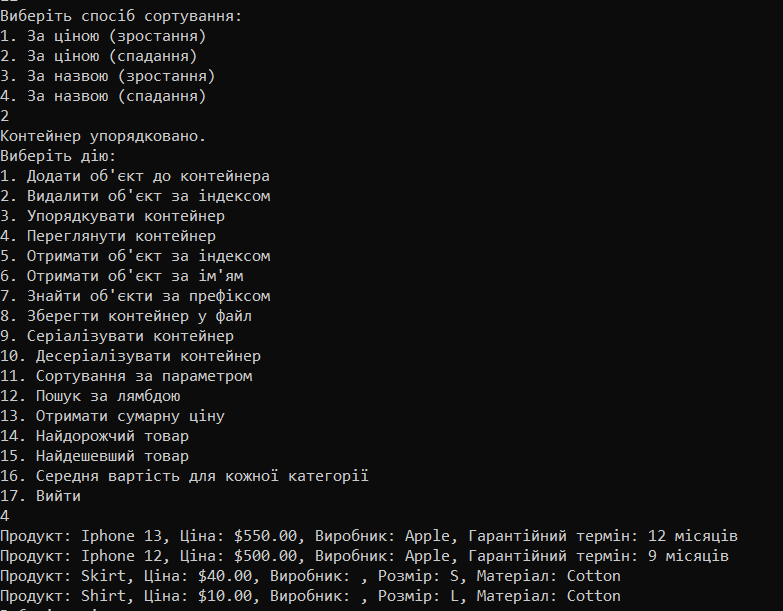


Рисунок 10. Сортування за параметром

Функція пошуку за лямбда-виразом у головній програмі:

static void FindItemsByCondition(Container<Product> container)

{

Console.WriteLine("Виберіть умову для пошуку:");

Console.WriteLine("1. Ціна більше заданого значення");

Console.WriteLine("2. Назва містить певний рядок");

string? choice = Console.ReadLine();

IEnumerable<Product> foundItems = Enumerable.Empty<Product>();

switch (choice)

{

case "1":

Console.Write("Введіть значення ціни: ");

string? priceInput = Console.ReadLine();

if (decimal.TryParse(priceInput, out decimal price))

{

foundItems = container.Find(item => item.Price > price);

}

else

{

Console.WriteLine("Невірний формат ціни.");

}

break;

case "2":

Console.Write("Введіть рядок для пошуку в назві: ");

string? substring = Console.ReadLine();

if (!string.IsNullOrEmpty(substring))

{

foundItems = container.Find(item => item.Name.Contains(substring, StringComparison.OrdinalIgnoreCase));

}

else

{

Console.WriteLine("Рядок не може бути порожнім.");

}

break;

default:

Console.WriteLine("Невірний вибір.");

break;

}

foreach (var item in foundItems)

{

Console.WriteLine(item);

}

}

Метод пошуку за лямбда виразом у контейнері:

public IEnumerable<T> Find(Func<T, bool> predicate)

{

try

{

return items.Where(predicate);

}

catch (Exception ex)

{

throw new Exception("Помилка пошуку елементів", ex);

}

}

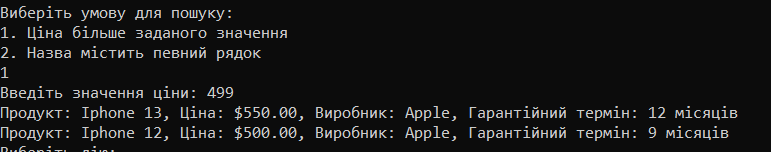


Рисунок 6. Пошук за лямбда виразом

# Отримання сумарної вартості

Функція в головній програмі:

static void GetTotalValue(Container<Product> container)

{

decimal totalValue = container.GetTotalValue();

Console.WriteLine($"Загальна вартість товарів на складі: {totalValue:C}");

}

Метод контейнера:

public decimal GetTotalValue()

{

return items.Sum(item => item.Price);

}



Рисунок 12. Сумарна вартість

# LINGQ

**Знаходження найдорожчого і найдешевшого товару**

Методи контейнеру:

public T FindCheapestItem()

{

if (!items.Any())

throw new InvalidOperationException("Контейнер порожній");

return items.OrderBy(item => item.Price).First();

}

public T FindMostExpensiveItem()

{

if (!items.Any())

throw new InvalidOperationException("Контейнер порожній");

return items.OrderByDescending(item => item.Price).First();

}

Функцій в головній програмі:  
static void FindMostExpensiveItem(Container<Product> container)

{

try

{

var mostExpensiveItem = container.FindMostExpensiveItem();

Console.WriteLine($"Найдорожчий товар: {mostExpensiveItem.Name}, Ціна: {mostExpensiveItem.Price:C}");

}

catch (InvalidOperationException ex)

{

Console.WriteLine(ex.Message);

}

}

static void FindCheapestProduct(Container<Product> container)

{

try

{

var cheapestProduct = container.FindCheapestItem();

Console.WriteLine("Найдешевший продукт: " + cheapestProduct);

}

catch (InvalidOperationException ex)

{

Console.WriteLine(ex.Message);

}

}





Рисунок 13. Найдорожчий і найдешевший товар

**Знаходження середньої вартості для кожної категорії**

Функція в головній програмі:

static void GetAveragePriceByCategory(Container<Product> container)

{

var averagePrices = container.GetAveragePriceByCategory();

foreach (var category in averagePrices)

{

Console.WriteLine($"Категорія: {category.Key}, Середня ціна: {category.Value:C}");

}

}

Метод контейнера:

public Dictionary<string, decimal> GetAveragePriceByCategory()

{

return items

.GroupBy(item => item.GetType().Name)

.ToDictionary(g => g.Key, g => g.Average(item => item.Price));

}

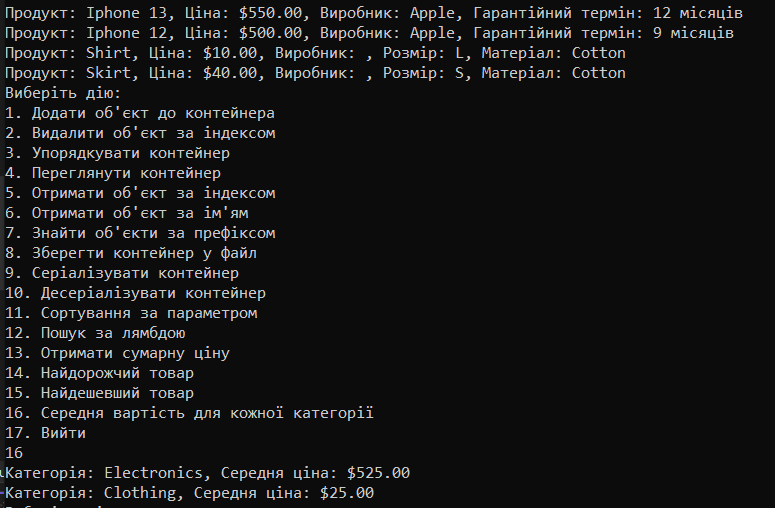


Рисунок 14. Середня ціна для категорій