**Боб Олександр**

**#1**

using System;

using System.Collections.Generic;

using System.Text;

using System.IO;

namespace HomeTask2.classes

{

public class Product

{

public Product(string \_name, int \_weight, double \_price, int \_expirationDate, DateTime \_creationTime)

{

this.Name = \_name;

this.Price = \_price;

this.Weight = \_weight;

this.expirationDate = \_expirationDate;

this.creationTime = \_creationTime;

}

public Product() : this(\_name: "NAME", \_weight: 0, \_price: 0, \_expirationDate: 1, \_creationTime: DateTime.Now) { }

public virtual void Parse(string dataForParse)

{

string[] initialisationUnits = dataForParse.Split(" ", StringSplitOptions.RemoveEmptyEntries);

if (initialisationUnits.Length != 5)

{

throw new Exception($"Impossible to initialise {this.GetType()} object - invalid string for parse");

}

this.Name = initialisationUnits[0];

try

{

this.Weight = Convert.ToInt32(initialisationUnits[1]);

}

catch (Exception ex)

{

throw new Exception($"Impossible to initialise {this.GetType()} object - invalid 'weight' parameter");

}

try

{

this.Price = Convert.ToDouble(initialisationUnits[2]);

}

catch (Exception ex)

{

throw new Exception($"Impossible to initialise {this.GetType()} object - invalid 'price' parameter");

}

try

{

this.ExpirationDate = Convert.ToInt32(initialisationUnits[3]);

}

catch (Exception ex)

{

throw new Exception($"Impossible to initialise {this.GetType()} object - invalid 'expiration date' parameter");

}

try

{

this.CreationTime = DateTime.Parse(initialisationUnits[4]);

}

catch (Exception ex)

{

throw new Exception($"Impossible to initialise {this.GetType()} object - invalid 'creation time' parameter");

}

}

int expirationDate;

public int ExpirationDate

{

get

{

return expirationDate;

}

set

{

if (value > 0)

{

expirationDate = value;

}

}

}

private DateTime creationTime;

public DateTime CreationTime

{

get { return creationTime; }

private set { creationTime = value; }

}

private string name;

public string Name

{

get { return name; }

protected set { name = value; }

}

//Weight is measured in grams

public int Weight

{

get;

set;

}

private double price;

public double Price

{

get

{

return price;

}

set

{

if (value < 0)

{

price = (-1) \* value;

}

else

{

price = value;

}

}

}

public virtual void ChangePrice(double Percentage, (int, int, int) Koefs)//Koefs - коефіцієнти, які будуть задаватися на складі.

{ // У випадку молочних або м'ясних продуктів вони впливатимуть на формування ціни

Price \*= 1 + (Percentage / 100);

Price = Math.Round(Price, 2);

}

public override bool Equals(Object obj)

{

if (this.GetType() == obj.GetType())

{

var Second = (Product)obj;

return this.Name == Second.Name && this.Price == Second.Price && this.Weight == Second.Weight;

}

return false;

}

public override string ToString()

{

return $"Name: \"{this.Name}\", Weight: {this.Weight}, Price:{this.Price}";

}

public Product Copy()

{

return (Product)this.MemberwiseClone();

}

}

}

using System;

using System.Collections.Generic;

using System.Text;

namespace HomeTask2.classes

{

enum Sort

{

Greatest = 0,

First = 1,

Second = 2

}

enum Type

{

Lamb, //Баранина

Veal, //Телятина

Pork, //Свинина

Chicken //Курятина

}

class Meat : Product

{

public Meat(string \_name, int \_weight, double \_price, int \_expirationDate, DateTime \_creationTime, Sort \_sort, Type \_type) :

base(\_name, \_weight, \_price, \_expirationDate, \_creationTime)

{

this.SortOfTheProduct = \_sort;

this.TypeOfTheProduct = \_type;

}

public Meat() : base() { this.SortOfTheProduct = Sort.Second; this.TypeOfTheProduct = Type.Chicken; }

public override void Parse(string dataForParse)

{

string[] initialisationUnits = dataForParse.Split();

try

{

base.Parse(initialisationUnits[0] + ' ' + initialisationUnits[1] + ' ' + initialisationUnits[2] + ' ' +

initialisationUnits[3] + ' ' + initialisationUnits[4]);

}

catch(Exception ex)

{

throw;// new Exception($"Impossible to initialise {this.GetType()} object - invalid string for parse");

}

try

{

this.SortOfTheProduct = (Sort)Enum.Parse(typeof(Sort), initialisationUnits[5]);

}

catch (Exception)

{

throw new Exception($"Impossible to initialise {this.GetType()} object - invalid 'sort parameter");

}

try

{

this.TypeOfTheProduct = (Type)Enum.Parse(typeof(Type), initialisationUnits[6]);

}

catch (Exception)

{

throw new Exception($"Impossible to initialise {this.GetType()} object - invalid 'sort parameter");

}

}

private Sort sortOfTheProduct;

public Sort SortOfTheProduct

{

get { return sortOfTheProduct; }

protected set

{

try

{

sortOfTheProduct = value;

}

catch (System.Exception e)

{

Console.WriteLine("Impossible to use this Sort.");

Console.WriteLine(e.Message);

}

}

}

private Type typeOfTheProduct;

public Type TypeOfTheProduct

{

get { return typeOfTheProduct; }

protected set

{

try

{

typeOfTheProduct = value;

}

catch (System.Exception e)

{

Console.WriteLine("Impossible to use this Sort.");

Console.WriteLine(e.Message);

}

}

}

public override void ChangePrice(double Percentage, (int, int, int) Coefs)

{

base.ChangePrice(Percentage, Coefs);

if (SortOfTheProduct == Sort.Greatest)

{

base.ChangePrice(Coefs.Item1, Coefs);

}

else if (SortOfTheProduct == Sort.First)

{

base.ChangePrice(Coefs.Item2, Coefs);

}

else if (SortOfTheProduct == Sort.Second)

{

base.ChangePrice(Coefs.Item3, Coefs);

}

}

public override bool Equals(Object obj)

{

if (this.GetType() == obj.GetType())

{

var Second = (Meat)obj;

return this.Name == Second.Name &&

this.Price == Second.Price &&

this.Weight == Second.Weight &&

this.SortOfTheProduct == Second.SortOfTheProduct &&

this.TypeOfTheProduct == Second.TypeOfTheProduct;

}

return false;

}

public override string ToString()

{

return $"Name: \"{this.Name}\", Weight: {this.Weight}, Price:{this.Price}\n" +

$"Sort: \"{this.SortOfTheProduct}\", Type: \"{this.TypeOfTheProduct}\"";

}

public new Meat Copy()

{

return (Meat)this.MemberwiseClone();

}

}

}

using System;

using System.Collections.Generic;

using System.Text;

namespace HomeTask2.classes

{

class Dairy\_products : Product

{

private int expirationDate;

public Dairy\_products(string \_name, int \_weight, double \_price, int \_expirationDate, DateTime \_creationTime) :

base(\_name, \_weight, \_price, \_expirationDate, \_creationTime){}

public Dairy\_products() :base(){ }

public override void Parse(string dataForParse)

{

string[] initialisationUnits = dataForParse.Split();

try

{

base.Parse(initialisationUnits[0] + ' ' + initialisationUnits[1] + ' ' + initialisationUnits[2] + ' ' +

initialisationUnits[3] + ' ' + initialisationUnits[4]);

}

catch

{

throw;// new Exception($"Impossible to initialise {this.GetType()} object - invalid string for parse");

}

}

public override void ChangePrice(double Percentage, (int, int, int) Coefs)

{

base.ChangePrice(Percentage, Coefs);

if (ExpirationDate < 7)

{

base.ChangePrice(Coefs.Item1, Coefs);

}

else if (ExpirationDate < 14 && ExpirationDate >= 7)

{

base.ChangePrice(Coefs.Item2, Coefs);

}

else if (ExpirationDate >= 14)

{

base.ChangePrice(Coefs.Item3, Coefs);

}

}

public override bool Equals(Object obj)

{

if (this.GetType() == obj.GetType())

{

var Second = (Dairy\_products)obj;

return this.Name == Second.Name &&

this.Price == Second.Price &&

this.Weight == Second.Weight &&

this.ExpirationDate == Second.ExpirationDate;

}

return false;

}

public override string ToString()

{

return $"Name: \"{this.Name}\", Weight: {this.Weight}, Price:{this.Price}\n" +

$"Expiration date: {this.ExpirationDate}";

}

public new Dairy\_products Copy()

{

return (Dairy\_products)this.MemberwiseClone();

}

}

}

using System;

using System.Collections.Generic;

using System.Text;

using System.IO;

namespace HomeTask2.classes

{

public class Storage

{

private Product[] products;

(int, int, int) Koefs;

bool IsInitialised = false;

public Storage() { }

public void ChangePrices(double \_percentage)

{

foreach (var product in products)

{

product.ChangePrice(Percentage: \_percentage, Koefs);

}

}

public Product this[int Index]

{

get { return products[Index]; }

set { products[Index] = value; }

}

public void GetFullInfo()

{

foreach (var product in products)

{

Console.WriteLine(product + "\n");

}

}

public Product[] GetAllMeatProducts()

{

Product[] meat;

int CountOfMeatProducts = 0;

for (int i = 0; i < products.Length; i++)

{

if (products[i] is Meat)

{

CountOfMeatProducts++;

}

}

meat = new Product[CountOfMeatProducts];

CountOfMeatProducts = 0;

for (int i = 0; i < products.Length; i++)

{

if (products[i] is Meat)

{

meat[CountOfMeatProducts++] = products[i];

}

}

return meat;

}

public void GetBadDairyProducts(string filePath)

{

string result = "";

Dairy\_products dairy = null;

try

{

using (StreamWriter sw = new StreamWriter(filePath))

{

for (int i = 0; i < products.Length; i++)

{

if ((dairy = products[i] as Dairy\_products) != null)

{

if ((DateTime.Today - dairy.CreationTime).TotalDays > dairy.ExpirationDate)

{

result += $"{dairy.Name} {dairy.Weight} {dairy.Price} {dairy.ExpirationDate} {dairy.CreationTime}\n";

products[i] = null;

}

}

}

sw.Write(result);

}

}

catch (Exception ex)

{

throw new Exception("Failed to get bad dairy products: " + ex.Message);

}

}

public void StartFileInitialisation(string filePath)

{

string[] initialisationParameters = null;

try

{

using (StreamReader sr = new StreamReader(filePath))

{

initialisationParameters = sr.ReadToEnd().Split('\n');

products = new Product[initialisationParameters.Length - 1];

try

{

string[] koefs = initialisationParameters[0].Split();

Koefs.Item1 = Convert.ToInt32(koefs[0]);

Koefs.Item2 = Convert.ToInt32(koefs[1]);

Koefs.Item3 = Convert.ToInt32(koefs[2]);

}

catch (Exception)

{

throw new Exception("Impossible to start initialisation of storage - invalid koefs");

}

string[] productInitialisationParameters = null;

try

{

for (int i = 1; i < initialisationParameters.Length; i++)

{

productInitialisationParameters = initialisationParameters[i].Split('|');

switch (productInitialisationParameters[0])

{

case ("Meat"):

products[i - 1] = new Meat();

//products[i - 1].Parse(productInitialisationParameters[1]);

break;

case ("Dairy"):

products[i - 1] = new Dairy\_products();

//products[i - 1].Parse(productInitialisationParameters[1]);

break;

case ("Classic"):

products[i - 1] = new Product();

break;

default:

throw new Exception("Impossible to start initialisation of instance in storage - invalid product type");

break;

}

products[i - 1].Parse(productInitialisationParameters[1]);

}

}

catch (Exception ex)

{

throw new Exception("Error occured on units initialisation stage: " + ex.Message);

}

}

}

catch (Exception ex)

{

throw new Exception("Storage initialisation from file failed: " + ex.Message);

}

IsInitialised = true;

}

public void StartFastInitialisation(int k1, int k2, int k3, params Product[] \_paramsProducts)

{

if (IsInitialised)

{

return;

}

products = new Product[\_paramsProducts.Length];

IsInitialised = true;

for (int i = 0; i < \_paramsProducts.Length; i++)

{

products[i] = \_paramsProducts[i].Copy();

}

Koefs = (k1, k2, k3);

}

public void StartSlowInitialisation()

{

if (IsInitialised)

{

return;

}

Console.WriteLine("How much elemests shoud your storage have?");

int Count = Convert.ToInt32(Console.ReadLine());

this.products = new Product[Count];

IsInitialised = true;

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

{

Console.WriteLine($"Prod #{i + 1}");

Console.WriteLine("Which type of product do you want to insert?\n1.Classic\n2.Meat\n3.Dairy\n");

string choice = Console.ReadLine();

string name;

int weight;

double price;

int expirationDate;

DateTime creationTime;

switch (choice)

{

case ("Classic"):

try

{

Console.WriteLine("Input name");

name = Console.ReadLine();

Console.WriteLine("Input weight");

weight = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Input price");

price = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Input expiration date");

expirationDate = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Input creation time");

creationTime = DateTime.Parse(Console.ReadLine());

products[i] = new Product(\_name: name, \_weight: weight, \_price: price, \_expirationDate: expirationDate, \_creationTime: creationTime);

}

catch (Exception exx)

{

throw new Exception("Impossible to initialise 'Product' object: " + exx.Message);

}

break;

case ("Meat"):

try

{

Console.WriteLine("Input name");

name = Console.ReadLine();

Console.WriteLine("Input weight");

weight = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Input price");

price = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Input expiration date");

expirationDate = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Input creation time");

creationTime = DateTime.Parse(Console.ReadLine());

Type t = 0;

Console.WriteLine("\nChoose type\nLamb\nVeal\nPork\nChicken\n");

switch (Console.ReadLine())

{

case ("Lamb"):

t = Type.Lamb;

break;

case ("Veal"):

t = Type.Veal;

break;

case ("Pork"):

t = Type.Pork;

break;

case ("Chicken"):

t = Type.Chicken;

break;

}

Console.WriteLine("\nChoose type\nGreatest\nFirst\nSecond\n");

Sort s = 0;

switch (Console.ReadLine())

{

case ("Greatest"):

s = Sort.Greatest;

break;

case ("First"):

s = Sort.First;

break;

case ("Second"):

s = Sort.Second;

break;

}

products[i] = new Meat(\_name: name, \_weight: weight, \_price: price, \_sort: s, \_type: t, \_expirationDate: expirationDate, \_creationTime: creationTime);

}

catch (Exception exx)

{

throw new Exception("Impossible to initialise 'meat' object: " + exx.Message);

}

break;

case ("Dairy"):

try

{

Console.WriteLine("Input name");

name = Console.ReadLine();

Console.WriteLine("Input weight");

weight = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Input price");

price = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Input expire date");

int ex = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Input expiration date");

expirationDate = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Input creation time");

creationTime = DateTime.Parse(Console.ReadLine());

products[i] = new Dairy\_products(\_name: name, \_weight: weight, \_price: price, \_expirationDate: expirationDate, \_creationTime: creationTime);

}

catch (Exception exx)

{

throw new Exception("Impossible to initialise 'dairy' object: " + exx.Message);

}

break;

default:

Console.WriteLine("Invalid input!");

i--;

continue;

}

}

int koef1, koef2, koef3;

Console.WriteLine("\nInput koef #1:");

koef1 = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("\nInput koef #2:");

koef2 = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("\nInput koef #3:");

koef3 = Convert.ToInt32(Console.ReadLine());

Koefs = (koef1, koef2, koef3);

}

}

}

**#2**

using System;

using System.Collections.Generic;

using System.Text;

using System.IO;

namespace Poly

{

class Polynomial

{

private double[] koefs = null;

private int m;

public int M

{

get { return m; }

private set { m = value; }

}

public Polynomial(string initString)

{

this.Parse(initString);

}

public Polynomial(int \_m)

{

this.M = \_m;

koefs = new double[M];

}

public Polynomial Multiply(Polynomial p2)

{

Polynomial result = new Polynomial(this.M + p2.M);

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

{

for (int j = 0; j < M; j++)

{

result.koefs[i + j] += koefs[i] \* p2.koefs[j];

}

}

return result;

}

public Polynomial Minus(Polynomial p2)

{

Polynomial result;

if (this.M >= p2.M)

{

result = new Polynomial(this.M);

}

else

{

result = new Polynomial(p2.M);

}

for (int i = 0; i < this.M; i++)

{

result.koefs[i] += this.koefs[i];

}

for (int i = 0; i < p2.M; i++)

{

result.koefs[i] -= p2.koefs[i];

}

return result;

}

public Polynomial Add(Polynomial p2)

{

Polynomial result;

if (this.M >= p2.M)

{

result = new Polynomial(this.M);

}

else

{

result = new Polynomial(p2.M);

}

for (int i = 0; i < this.M; i++)

{

result.koefs[i] += this.koefs[i];

}

for (int i = 0; i < p2.M; i++)

{

result.koefs[i] += p2.koefs[i];

}

return result;

}

public double this[int index]

{

get

{

return koefs[index];

}

set

{

if (index < 0)

{

throw new IndexOutOfRangeException();

}

if (value != 0)

{

if (index >= M)

{

double[] temp = new double[index + 1];

for (int i = 0; i < koefs.Length; i++)

{

temp[i] = koefs[i];

}

koefs = temp;

M = index + 1;

}

koefs[index] = value;

}

else

{

if (index < M)

{

koefs[index] = value;

}

}

}

}

public double GetValue(double val)

{

double result = 0;

for (int i = 0; i < koefs.Length; i++)

{

result += koefs[i] \* Math.Pow(val, i);

}

return result;

}

public void Parse(string inputStr)

{

string[] polyElements = inputStr.Split("+", StringSplitOptions.RemoveEmptyEntries);

try

{

M = Convert.ToInt32(polyElements[polyElements.Length - 1].Split("^")[1]);

}

catch (Exception)

{

throw new Exception("Impossible to initialise polynom with this parameters");

}

koefs = new double[++M];

try

{

for (int i = 0; i < polyElements.Length; i++)

{

koefs[Convert.ToInt32(polyElements[i].Split("^")[1])] = Convert.ToDouble(polyElements[i].Split("\*")[0]);

}

}

catch

{

throw new Exception("Invalid parameter in file - impossible to convert");

}

}

}

}