// Product.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Task2

{

public class Product

{

private string name = "Unnamed";

private double cost = 0.0;

private double weight = 0.0;

public string Name

{

get => name;

set

{

if (value == null)

{

throw new ArgumentNullException(nameof(value), $"{nameof(Name)} cannot be set to null.");

}

if (string.Compare(value, "") == 0)

{

throw new ArgumentException($"{nameof(value)} is empty string.", $"{nameof(Name)} cannot be set to empty string.");

}

name = value;

}

}

public double Cost

{

get => cost;

set

{

if (value <= 0.0)

{

throw new ArgumentException($"{nameof(value)} is lower than 0.0", $"{nameof(Cost)} cannot be set to negative number.");

}

cost = value;

}

}

public double Weight

{

get => weight;

set

{

if (value < 0.0)

{

throw new ArgumentException($"{nameof(value)} is lower than 0.0", $"{nameof(Weight)} cannot be set to negative number.");

}

weight = value;

}

}

public Product(string name, double cost, double weight)

{

this.Name = name;

this.Cost = cost;

this.Weight = weight;

}

public Product(Product product) : this(product.Name, product.Cost, product.Weight) { }

public virtual void ChangeCost(double percents)

{

if (percents < -100.0)

{

throw new ArgumentException($"Invalid percents passed. {nameof(percents)} must be greater or equal to -100");

}

double change = Cost \* percents / 100.0;

Cost += change;

}

public override bool Equals(object obj)

{

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

{

return false;

}

var other = obj as Product;

return String.Compare(this.Name, other.name) == 0 &&

this.Cost == other.Cost &&

this.Weight == other.Weight;

}

public override string ToString()

{

return String.Concat($"Name: {this.Name}\n", $"Cost: ${this.Cost}\n", $"Weight: {this.Weight} kg");

}

public override int GetHashCode()

{

return HashCode.Combine(this.Name, this.Cost, this.Weight);

}

}

}

// Category.cs

namespace Task2

{

public enum Category

{

Highest,

First,

Second

}

}

// Kind.cs

namespace Task2

{

public enum Kind

{

Sheep,

Beef,

Pork,

Chicken

}

}

// Meat.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Task2

{

public class Meat : Product

{

public Category Category { get; set; }

public Kind Kind { get; set; }

public Meat(string name, double cost, double weight, Category category, Kind kind) : base(name, cost, weight)

{

Category = category;

Kind = kind;

}

public Meat(Meat meat) : base(meat)

{

Category = meat.Category;

Kind = meat.Kind;

}

public override void ChangeCost(double percents)

{

if (percents < -100.0)

{

throw new ArgumentException($"Invalid percents passed. {nameof(percents)} must be greater or equal to -100");

}

base.ChangeCost(percents);

switch (this.Category)

{

case Category.First:

base.ChangeCost(-5.0);

break;

case Category.Second:

base.ChangeCost(-10.0);

break;

default:

break;

}

}

public override bool Equals(object obj)

{

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

{

return false;

}

var other = obj as Meat;

return this.Category.Equals(other.Category) &&

this.Kind.Equals(other.Kind) &&

base.Equals(obj);

}

public override string ToString()

{

return String.Concat($"Category: {this.Category.ToString()}\n", $"Kind: {this.Kind.ToString()}\n", base.ToString());

}

public override int GetHashCode()

{

return HashCode.Combine(this.Category, this.Kind, base.GetHashCode());

}

}

}

// DairyProducts.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Task2

{

public class DairyProducts : Product

{

private int daysToExpiration = 365;

public int DaysToExpiration

{

get => daysToExpiration;

set

{

if (value < 0)

{

throw new ArgumentException($"{nameof(value)} is lower than 0", $"{nameof(DaysToExpiration)} cannot be set to negative number.");

}

daysToExpiration = value;

}

}

public DairyProducts(string name, double cost, double weight, int daysToExpiration) : base(name, cost, weight)

{

this.DaysToExpiration = daysToExpiration;

}

public DairyProducts(DairyProducts dairyProducts) : base(dairyProducts)

{

this.DaysToExpiration = dairyProducts.DaysToExpiration;

}

public override void ChangeCost(double percents)

{

if (percents < -100.0)

{

throw new ArgumentException($"Invalid percents passed. {nameof(percents)} must be greater or equal to -100");

}

base.ChangeCost(percents);

switch (this.DaysToExpiration)

{

case <= 5:

base.ChangeCost(-20.0);

break;

case <= 30:

base.ChangeCost(-8.0);

break;

default:

break;

}

}

public override bool Equals(object obj)

{

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

{

return false;

}

var other = obj as DairyProducts;

return this.DaysToExpiration == other.DaysToExpiration &&

base.Equals(other);

}

public override string ToString()

{

return String.Concat($"Days to expiration: {this.DaysToExpiration}\n", base.ToString());

}

public override int GetHashCode()

{

return HashCode.Combine(DaysToExpiration, base.GetHashCode());

}

}

}

// Buy.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Task2

{

public class Buy

{

private List<Product> products;

public List<Product> Products

{

get => products;

}

public int Count

{

get => Products.Count;

}

public double OverallCost

{

get => Products.Sum(product => product.Cost);

}

public double OverallWeight

{

get => Products.Sum(product => product.Weight);

}

public Buy(params Product[] products)

{

this.products = new List<Product>();

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

{

if (products[i] == null)

{

throw new ArgumentNullException($"products[{i}] is null.");

}

Products.Add(new Product(products[i]));

}

}

public override string ToString()

{

StringBuilder sb = new();

sb.Append($"Count: {Count}\n");

sb.Append(String.Join("\n", Products.Select((product, index) => $"Product #{index + 1}:\n{product.ToString()}")));

sb.Append($"Overall weight: {OverallWeight}\n");

sb.Append($"Overall cost: {OverallCost}\n");

return sb.ToString();

}

public override bool Equals(object obj)

{

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

{

return false;

}

var other = obj as Buy;

return this.Count == other.Count &&

this.OverallWeight == other.OverallWeight &&

this.OverallCost == other.OverallCost &&

this.Products.Equals(other.Products);

}

public override int GetHashCode()

{

return HashCode.Combine(this.Count, this.OverallCost, this.OverallWeight, this.Products);

}

}

}

// Check.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Task2

{

public sealed class Check // other classes cannot inherit this Check class anymore.

{

private Check() { }

public static string PrintProduct(Product product)

{

if (product == null)

{

throw new ArgumentNullException(nameof(product));

}

return product.ToString();

}

public static string PrintBuy(Buy buy)

{

if (buy == null)

{

throw new ArgumentNullException(nameof(buy));

}

return buy.ToString();

}

}

}

// Storage.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Task2

{

public class Storage

{

private Product[] products = null;

public Product this[int index]

{

get

{

if (products == null || index < 0 || index >= products.Length)

{

throw new IndexOutOfRangeException($"Index was out of range. Must be non-negative and less than the size of the array. Parameter name: {nameof(index)}");

}

return products[index];

}

}

public int Length

{

get => products.Length;

}

public void ReadProducts(int count)

{

if (count < 1)

{

throw new ArgumentException($"{nameof(count)} is lower than 1.");

}

if (products == null || products.Length != count)

{

products = new Product[count];

}

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

{

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

Console.Write("Select type: Product(1), Meat(2), DairyProduct(3): ");

int chose = Int32.Parse(Console.ReadLine());

string name;

double cost, weight;

Console.Write("Name: ");

name = Console.ReadLine();

Console.Write("Cost: ");

cost = Double.Parse(Console.ReadLine());

Console.Write("Weight: ");

weight = Double.Parse(Console.ReadLine());

switch (chose)

{

case 2:

Console.Write("Category: Highest (0), First (1), Second(2): ");

Category category = (Category)Enum.Parse(typeof(Category), Console.ReadLine());

Console.Write("Kind: Sheep(0), Beef(1), Pork(2), Chicken(3): ");

Kind kind = (Kind)Enum.Parse(typeof(Kind), Console.ReadLine());

products[i] = new Meat(name, cost, weight, category, kind);

break;

case 3:

Console.Write("Days to expiration: ");

int daysToExpiration = Int32.Parse(Console.ReadLine());

products[i] = new DairyProducts(name, cost, weight, daysToExpiration);

break;

default:

products[i] = new Product(name, cost, weight);

break;

}

}

}

public string PrintProducts()

{

StringBuilder sb = new();

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

{

sb.Append($"Product #{i + 1}\n");

sb.Append(Check.PrintProduct(products[i]) + "\n");

}

return sb.ToString();

}

public Meat[] GetAllMeatProducts()

{

return products.Where(prod => prod as Meat != null).Select(meat => meat as Meat).ToArray();

}

public void ChangeCostForAll(double percents)

{

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

{

products[i].ChangeCost(percents);

}

}

public Storage(params Product[] products)

{

this.products = new Product[products.Length];

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

{

if (products[i] == null)

{

throw new ArgumentNullException($"products[{i}] is null");

}

this.products[i] = new Product(products[i]);

}

}

public Storage(int count)

{

products = new Product[count];

ReadProducts(count);

}

}

}

// Program.cs

using System;

namespace Task2

{

class Program

{

static void Main(string[] args)

{

DemonstrateMethods();

}

static void DemonstrateMethods()

{

Storage storage = new(3);

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

{

storage.ChangeCostForAll(10);

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

Console.WriteLine(Check.PrintProduct(storage[i]));

}

Storage storage1 = new(

new Meat("Chicken leg", 43.5, 6, Category.First, Kind.Chicken),

new Product("NVidia RTX 3090", 799, 0.8),

new DairyProducts("Bread", 1.0, 0.8, 2)

);

Console.WriteLine(storage1.PrintProducts());

Meat[] meats = storage1.GetAllMeatProducts();

foreach (Meat meat in meats)

{

Console.WriteLine(Check.PrintProduct(meat));

}

}

}

}