**FILE PROGRAM.CS**

using System;

namespace hw2

{

public class Program

{

static void Main(string[] args)

{

try

{

Product Processor = new("Intel I9-9900K", 449.90, 0.10);

Meat Beef = new Meat("Beef meat", 20, 20, MeatCategory.First, MeatSort.Beef);

Buy ProcessorPurchase = new(Processor, 20);

DairyProducts Dairy = new DairyProducts("milk", 20, 1, 30);

Storage Assortment = new Storage(Processor, Beef, Dairy);

System.Console.WriteLine(Assortment.ToString());

Storage SearchMeatResult = Assortment.SearchForMeat();

System.Console.WriteLine(SearchMeatResult.ToString());

}

catch (ArgumentException ex)

{

Console.WriteLine("\nError occured:\n" + ex.Message + "\nExiting programm");

return;

}

}

}

}

**FILE PRODUCT.CS**

using System;

using System.Text;

namespace hw2

{

public class Product

{

private string \_name;

private double \_price;

private double \_weight;

public string Name

{

get => \_name;

set

{

if (String.Compare(value, "") != 0)

{

\_name = value;

}

else

{

throw new ArgumentException("Name can't be empty");

}

}

}

public double Price

{

get => \_price;

set

{

if (value >= 0)

{

\_price = value;

}

else

{

throw new ArgumentException("Price must be a positive number");

}

}

}

public double Weight

{

get => \_weight;

set

{

if (value > 0)

{

\_weight = value;

}

else

{

throw new ArgumentException("Weight must be a positive number");

}

}

}

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

{

Name = name;

Price = price;

Weight = weight;

}

public virtual void ChangePrice(double diff)

{

if ((diff.CompareTo(-1d) == -1) || (diff.CompareTo(1d) == 1))

{

throw new ArgumentException("Difference must be a number between -1 and 1");

}

Price += Price \* diff;

}

public override string ToString()

{

StringBuilder result = new StringBuilder();

result.Append("Name: " + \_name + "\n");

result.Append("Price: " + \_price.ToString() + "\n");

result.Append("Weight: " + \_weight.ToString() + "\n");

return result.ToString();

}

public override int GetHashCode()

{

return (Convert.ToInt32(\_price) << 2) ^ Convert.ToInt32(\_weight);

}

public override bool Equals(object obj)

{

if ((obj == null) || !(this.GetType().Equals(obj.GetType())))

{

return false;

}

else

{

Product temp = obj as Product;

return (this.Name.Equals(temp.Name) && (this.Price == temp.Price) && (this.Weight == temp.Weight));

}

}

}

}

**FILE BUY.CS**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace hw2

{

public class Buy

{

private List<Product> \_purchaseList;

private int \_amount;

private double \_totalPrice;

private double \_totalWeight;

public Buy(){

\_purchaseList = new List<Product>();

\_purchaseList.Add(new Product("Default", 1, 1));

Amount = 1;

\_totalPrice = \_amount \* this.\_purchaseList[0].Price;

\_totalWeight = \_amount \* this.\_purchaseList[0].Weight;

}

public Buy(Product purchase, int amount)

{

\_purchaseList = new List<Product>();

\_purchaseList.Add(purchase);

Amount = amount;

\_totalPrice = \_amount \* this.\_purchaseList[0].Price;

\_totalWeight = \_amount \* this.\_purchaseList[0].Weight;

}

public Buy(params Product[] products)

{

\_purchaseList = new List<Product>();

Amount = products.Length;

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

{

if (products[i] == null)

{

throw new ArgumentException("Invalid agument");

}

else

{

\_purchaseList.Add(products[i]);

}

}

\_totalPrice = \_purchaseList.Sum(product => product.Price);

\_totalWeight = \_purchaseList.Sum(product => product.Weight);

}

public List<Product> PurchaseList

{

get => \_purchaseList;

}

public int Amount

{

get => \_amount;

set

{

if (value > 0)

{

\_amount = value;

}

else

{

throw new ArgumentException("Amount must be a positive number");

}

}

}

public double TotalPrice { get => \_totalPrice; }

public double TotalWeight { get => \_totalWeight; }

public override string ToString()

{

var result = new StringBuilder();

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

{

result.Append("\n" + PurchaseList[i].ToString());

}

result.Append($"\n\nAmount: {Amount.ToString() } pcs");

result.Append($"\nTotal Price: {TotalPrice.ToString("$0.00")}");

result.Append($"\nTotal Weight: {TotalWeight.ToString("0.00 kg")}\n");

return result.ToString();

}

public override int GetHashCode()

{

return ((Convert.ToInt32(\_totalPrice + \_totalWeight) << 2) ^ \_amount);

}

public override bool Equals(object obj)

{

if ((obj == null) || !(this.GetType().Equals(obj.GetType())))

{

return false;

}

else

{

Buy temp = obj as Buy;

bool bTemp = true;

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

bTemp = bTemp && this.\_purchaseList[i].Equals(temp.\_purchaseList[i]);

}

return ( bTemp &&

(this.Amount == temp.Amount) &&

(this.TotalPrice == temp.TotalPrice) &&

(this.TotalWeight == temp.TotalWeight));

}

}

}

}

**FILE MEAT.CS**

using System;

using System.Text;

namespace hw2

{

public class Meat : Product

{

private MeatCategory \_category;

private MeatSort \_sort;

public MeatCategory Category

{

get => \_category;

set

{

if (Enum.IsDefined(typeof(MeatCategory), value))

{

\_category = value;

}

}

}

public MeatSort Sort

{

get => \_sort;

set

{

if (Enum.IsDefined(typeof(MeatSort), value))

{

\_sort = value;

}

}

}

public Meat(string name, double price, double weight, MeatCategory category, MeatSort sort) : base(name, price, weight)

{

Category = category;

Sort = sort;

}

public override void ChangePrice(double diff)

{

base.ChangePrice(diff);

switch (Category)

{

case MeatCategory.High: base.ChangePrice(0.06); break;

case MeatCategory.First: base.ChangePrice(0.01); break;

case MeatCategory.Second: base.ChangePrice(-0.01); break;

}

}

public override string ToString()

{

StringBuilder result = new StringBuilder();

result.Append(base.ToString());

result.Append("Category: " + \_category + "\n");

result.Append("Sort: " + \_sort + "\n");

return result.ToString();

}

public override int GetHashCode()

{

return base.GetHashCode() << 2;

}

public override bool Equals(object obj)

{

if ((obj == null) || !(this.GetType().Equals(obj.GetType())))

{

return false;

}

else

{

Meat temp = obj as Meat;

return (base.Equals((Product)obj) && this.Category.Equals(temp.Category) && this.Sort.Equals(temp.Sort));

}

}

}

}

**FILE DAIRYPRODUCTS.CS**

using System;

using System.Text;

namespace hw2

{

public class DairyProducts : Product

{

private int \_expiration;

public int Expiration

{

get => \_expiration;

set

{

if (value > 0)

{

\_expiration = value;

}

else

{

throw new ArgumentException("Number of days to expiration must be a positive number");

}

}

}

public DairyProducts(string name, double price, double weight, int expiration) : base(name, price, weight)

{

Expiration = expiration;

}

public override void ChangePrice(double diff)

{

if ((diff.CompareTo(-1d) == -1) || (diff.CompareTo(1d) == 1))

{

throw new ArgumentException("Difference must be a number between -1 and 1");

}

base.ChangePrice(diff);

if (Expiration < 10)

{

base.ChangePrice(-0.3);

}

else if (10 <= Expiration && Expiration < 20)

{

base.ChangePrice(-0.1);

}

}

public override string ToString()

{

StringBuilder result = new StringBuilder();

result.Append(base.ToString());

result.Append("Expiration date: " + \_expiration.ToString() + "\n");

return result.ToString();

}

public override int GetHashCode()

{

return (base.GetHashCode() << 2) ^ \_expiration;

}

public override bool Equals(object obj)

{

if ((obj == null) || !(this.GetType().Equals(obj.GetType())))

{

return false;

}

else

{

DairyProducts temp = obj as DairyProducts;

return (base.Equals((Product)obj) && (this.Expiration == temp.Expiration));

}

}

}

}

**FILE MEATCATEGORY.CS**

namespace hw2

{

public enum MeatCategory { High = 1, First = 2, Second = 3 };

}

**FILE MEATSORT.CS**

namespace hw2

{

public enum MeatSort { Mutton = 1, Beef = 2, Pork = 3, Chicken = 4 };

}

**FILE STORAGE.CS**

using System;

using System.Text;

namespace hw2

{

public class Storage

{

private Product[] \_assortment;

public Storage(int size)

{

if (size < 0)

{

throw new ArgumentException("Size of array can't be a negative number");

}

\_assortment = new Product[size];

}

public Storage(params Product[] productArray)

{

\_assortment = new Product[productArray.Length];

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

{

\_assortment[i] = productArray[i];

}

}

public Product this[int index]

{

get

{

if (index < 0 || index > \_assortment.Length)

{

throw new ArgumentException("Index was out of dounds of array");

}

return \_assortment[index];

}

set

{

if (index < 0 || index > \_assortment.Length)

{

throw new ArgumentException("Index was out of dounds of array");

}

\_assortment[index] = value;

}

}

public override string ToString()

{

StringBuilder result = new StringBuilder();

if (\_assortment.Length == 0)

{

result.Append("Storage is empty");

return result.ToString();

}

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

{

result.Append(\_assortment[i].ToString() + "\n");

}

return result.ToString();

}

public Storage SearchForMeat()

{

int counter = 0;

foreach (var p in \_assortment)

{

if (p.GetType() == typeof(Meat))

{

++counter;

}

}

if (counter == 0)

{

return new Storage(0);

}

Storage temp = new Storage(counter);

for (int i = 0, ind = 0; i < \_assortment.Length; ++i)

{

if (\_assortment[i].GetType() == typeof(Meat))

{

temp[ind] = \_assortment[i];

++ind;

}

}

return temp;

}

public void ChangePrice(double diff)

{

if ((diff.CompareTo(-1d) == -1) || (diff.CompareTo(1d) == 1))

{

throw new ArgumentException("Difference must be a number between -1 and 1");

}

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

{

\_assortment[i].ChangePrice(diff);

}

}

public void ReadFromConsole(int size)

{

if (size < 1)

{

throw new ArgumentException("Size of strorage must be 1 and greater");

}

Storage temp = new Storage(size);

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

{

Console.WriteLine("Choose type of products:\n 1) Product\n 2) Meat\n 3)Dairy Product\n");

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

string name;

double price, weight;

Console.WriteLine("Enter name of product: ");

name = Console.ReadLine();

Console.WriteLine("Enter price of product: ");

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

Console.WriteLine("Enter weight of product: ");

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

switch (type)

{

case 2:

Console.WriteLine("Choose category 1)High 2)First 3)Second ");

MeatCategory mCategory = (MeatCategory)Enum.Parse(typeof(MeatCategory), Console.ReadLine());

Console.WriteLine("Choose meat sort: 1)Mutton 2)Beef 3)Pork 4)CHicken ");

MeatSort mSort = (MeatSort)Enum.Parse(typeof(MeatSort), Console.ReadLine());

\_assortment[i] = new Meat(name, price, weight, mCategory, mSort);

break;

case 3:

Console.WriteLine("Enter days before expiration: ");

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

\_assortment[i] = new DairyProducts(name, price, weight, expiration);

break;

default:

\_assortment[i] = new Product(name, price, weight);

break;

}

}

}

}

}

**FILE CHECK.CS**

using System.Text;

namespace hw2

{

sealed public class Check

{

public static string PrintBuy(Buy buy)

{

var result = new StringBuilder();

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

{

result.Append("\n" + PrintProduct(buy.PurchaseList[i]));

}

result.Append($"\n\nAmount: {buy.Amount.ToString() } pcs");

result.Append($"\nTotal Price: {buy.TotalPrice.ToString("$0.00")}");

result.Append($"\nTotal Weight: {buy.TotalWeight.ToString("0.00 kg")}\n");

return result.ToString();

}

public static string PrintProduct(Product product)

{

var result = new StringBuilder();

result.Append($"\nName: {product.Name}");

result.Append($"\nPrice: {product.Price.ToString("$0.00")}");

result.Append($"\nWeight: {product.Weight.ToString("0.00 kg")}");

return result.ToString();

}

}

}

**FILE CHECKINHERITED.CS**

namespace hw2

{

public class CheckInherited //: Check // IDE shows a problem: Cannot derive from sealed type 'Check'

{

};

}