|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Пермский филиал федерального государственного автономного образовательного учреждения высшего образования “Национальный исследовательский университет “Высшая школа экономики” | | | | |
|  | |  | |  |
|  | | *Факультет экономики, менеджмента и бизнес-информатики* | |  |
|  | |  | |  |
|  | | Дмитриев Арсений Алексеевич | |  |
|  | |  | |  |
| Лабораторная работа №10. «Наследование и виртуальные функции»  *Отчет по практической работе* | | | | |
|  | |  | |  |
|  | студента образовательной программы «Программная инженерия»  по направлению подготовки 09.03.04 Программная инженерия | | |  |
|  | |  | |  |
|  | |  | руководитель  к. тех. н., доцент каф.  инф. техн. в биз.  Викентьева О.Л. | |
|  | |  | |  |
|  | | Пермь, 2020 г. | |  |

Постановка задачи

Общая

Часть 1.

1. Определить иерархию классов (в соответствии с вариантом).
2. Реализовать классы.
3. Написать демонстрационную программу, в которой создаются объекты различных классов и помещаются в массив, после чего массив просматривается.
4. Реализовать 2 варианта программы: с помощью виртуальных и не виртуальных методов.
5. Объяснить почему без виртуальных функций программа будет работать неправильно. Объяснить необходимость виртуальных функций

Часть 2.

1. Реализовать метод для выполнения заданных запросов. При необходимости (для выполнения запроса) в класс могут быть добавлены новые поля (по сравнению с частью 1). В программе должно быть минимум ввода с клавиатуры. Поля объектов задаются в тексте программы. С клавиатуры вводятся только параметры запроса.
2. Реализовать не менее трех запросов, соответствующих иерархии классов (можно реализовать свои запросы).

Часть 3.

1. Составить иерархию классов в соответствии с вариантом. Во главе иерархии классов одолжен находиться интерфейс, которые определяет поведение объектов из иерархии классов (IExecutable).
2. Создать массив элементов типа IExecutable и поместить в него экземпляры различных классов иерархии. Выполнить просмотр массива, показать работу методов интерфейса IExecutable.
3. Реализовать сортировку элементов массива, используя стандартный интерфейс IComparable и метод Sort класса Array.
4. Реализовать сортировку и поиск элемента в массиве, используя стандартный интерфейс ICompare и метод Sort класса Array.
5. Реализовать метод клонирования объектов из интерфейса IClonable. Показать разницу между клонированием и поверхностным копированием объектов.

Вариант №7

Часть 1.

Иерархия: товар - > игрушка, продукт -> молочный продукт.

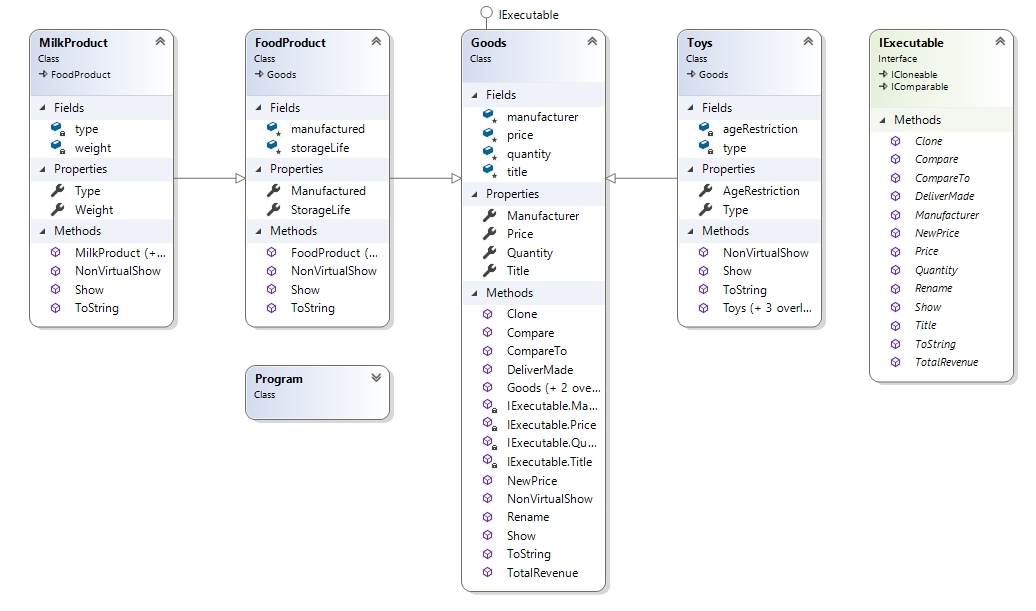
Часть 2

* 1. Суммарная стоимость товаров определенного производителя
  2. Все виды игрушек, для возрастов больше заданного
  3. Оставшееся количество молочной продукции определенного типа

Часть 3.

Сортировка и сравнение по общей стоимости товаров.

Диаграмма классов



Листинг программы

* + 1. Class Goods

public interface IExecutable : ICloneable, IComparable

{

int CompareTo(object obj);

int Compare(object obj1, object obj2);

object Clone();

string ToString();

int TotalRevenue();

void Show();

void Rename(string rename);

void NewPrice(int price);

void DeliverMade(int quantity);

string Title();

string Manufacturer();

int Price();

int Quantity();

}

public class Goods : IExecutable

{

protected int price;

protected int quantity;

protected string manufacturer;

protected string title;

public int Price

{

get { return price; }

protected set {

if (value < 0) price = 0;

else price = value;

}

}

public int Quantity

{

get { return quantity; }

protected set

{

if (value < 0) quantity = 0;

else quantity = value;

}

}

public string Manufacturer

{

get { return manufacturer; }

protected set { manufacturer = value; }

}

public string Title

{

get { return title; }

protected set { title = value; }

}

public Goods()

{

Title = "NotDetermined";

Manufacturer = "NotDetermined";

Price = 0;

Quantity = 0;

}

public Goods(string title, string manuf)

{

Title = title;

Manufacturer = manuf;

Price = 0;

Quantity = 0;

}

public Goods(string title, string manuf, int price, int quantity)

{

Title = title;

Manufacturer = manuf;

Price = price;

Quantity = quantity;

}

public void Rename(string rename)

{

Title = rename;

}

public void NewPrice(int price)

{

Price = price;

}

public void DeliverMade(int quantity)

{

Quantity = quantity;

}

public int TotalRevenue()

{

return Price \* Quantity;

}

public virtual void Show()

{

Console.WriteLine($"Товар:\n" +

$"Название: {Title}, производитель: {Manufacturer}\n" +

$"Стоимость: {Price}р., Количество на складе: {Quantity} шт.\n");

}

public void NonVirtualShow()

{

Console.WriteLine($"No virtual -> works as goods");

}

public override string ToString()

{

return $"Товар: {Title}, {Manufacturer}, {Price}р., {Quantity}";

}

#region Interface\_Options

public int CompareTo(object obj)

{

Goods good = (Goods)obj;

return good.TotalRevenue() - this.TotalRevenue();

}

public int Compare(object obj1, object obj2)

{

Goods good1 = (Goods)obj1;

Goods good2 = (Goods)obj2;

return good1.TotalRevenue() - good2.TotalRevenue();

}

public object Clone()

{

return new Goods(Title, Manufacturer, Price, Quantity);

}

string IExecutable.Title()

{

return Title;

}

string IExecutable.Manufacturer()

{

return Manufacturer;

}

int IExecutable.Price()

{

return Price;

}

int IExecutable.Quantity()

{

return Quantity;

}

#endregion

}

* + 1. Class Toys

public class Toys : Goods

{

private int ageRestriction;

private string type;

public int AgeRestriction

{

get { return ageRestriction; }

protected set {

if (value < 0) ageRestriction = 0;

else ageRestriction = value;

}

}

public string Type

{

get { return type; }

protected set { type = value; }

}

public Toys() : base()

{

AgeRestriction = 3;

Type = "NotDetermined";

}

public Toys(string title, string manuf) : base(title, manuf)

{

AgeRestriction = 3;

Type = "NotDetermined";

}

public Toys(string title, string manuf, int age, string type) : base(title, manuf)

{

AgeRestriction = age;

Type = type;

}

public Toys(string title, string manuf, int price, int quantity, int age, string type) : base(title, manuf, price, quantity)

{

AgeRestriction = age;

Type = type;

}

public override void Show()

{

Console.WriteLine($"Игрушка:\n" +

$"Название: {Title}, производитель: {Manufacturer}\n" +

$"Тип: {Type}, {AgeRestriction}+\n" +

$"Стоимость: {Price}р., Количество на складе: {Quantity} шт.\n");

}

public override string ToString()

{

return $"Игрушка: {Title}, {Manufacturer}, {Type}, {AgeRestriction}+, {Price}р., {Quantity} шт.";

}

public void NonVirtualShow()

{

Console.WriteLine($"No virtual -> works as toys");

}

}

* + 1. Class FoodProduct

public class FoodProduct : Goods

{

protected DateTime manufactured;

protected int storageLife;

public DateTime Manufactured

{

get { return manufactured; }

protected set { manufactured = value; }

}

public int StorageLife

{

get { return storageLife; }

protected set {

if (value < 0) storageLife = 0;

else storageLife = value;

}

}

public FoodProduct() : base()

{

Manufactured = DateTime.Today - (new TimeSpan(5, 12, 30, 0));

StorageLife = 168;

}

public FoodProduct(string title, string manuf) : base(title, manuf)

{

Manufactured = DateTime.Today - (new TimeSpan(5, 12, 30, 0));

StorageLife = 168;

}

public FoodProduct(string title, string manuf, int price, int quantity) : base(title, manuf, price, quantity)

{

Manufactured = DateTime.Today - (new TimeSpan(5, 12, 30, 0));

StorageLife = 168;

}

public FoodProduct(string title, string manufacturer, DateTime date, int life) : base(title, manufacturer)

{

Manufactured = date;

StorageLife = life;

}

public FoodProduct(string title, string manufacturer, int price, int quantity, DateTime date, int lifeSpan) : base(title, manufacturer, price, quantity)

{

Manufactured = date;

StorageLife = lifeSpan;

}

public override void Show()

{

Console.WriteLine($"Продукт питания:\n" +

$"Название: {Title}, производитель: {Manufacturer}\n" +

$"Произведено: {Manufactured}, срок годности: {StorageLife}ч\n" +

$"Стоимость: {Price}р., Количество на складе: {Quantity} шт.\n");

}

public void NonVirtualShow()

{

Console.WriteLine($"No virtual -> works as foods");

}

public override string ToString()

{

return $"Продукт: {Title}, {Manufacturer}, {Manufactured}, {storageLife} ч, {Price}р., {Quantity} шт.";

}

}

* 1. Class MilkProduct

public class MilkProduct : FoodProduct

{

private string type;

private double weight;

public string Type

{

get { return type; }

protected set { type = value;}

}

public double Weight {

get { return weight; }

protected set

{

if (weight < 0) weight = 0;

else weight = value;

}

}

public MilkProduct() : base()

{

Type = "Undetermined";

Weight = 1.0;

}

public MilkProduct(string title, string manuf) : base(title, manuf)

{

Type = "Undetermined";

Weight = 1.0;

}

public MilkProduct(string title, string manuf, string type, double weight) : base (title, manuf)

{

Type = type;

Weight = weight;

}

public MilkProduct(string title, string manuf, int price, int quantity, string type, double weight) : base(title, manuf, price, quantity)

{

Type = type;

Weight = weight;

}

public MilkProduct(string title, string manuf, int price, int quantity, DateTime date, int lifeSpan, string type, double weight) : base(title, manuf, price, quantity, date, lifeSpan)

{

Type = type;

Weight = weight;

}

public override void Show()

{

Console.WriteLine($"Молочные продукты:\n" +

$"Название: {Title}, производитель: {Manufacturer}\n" +

$"Произведено: {Manufactured}, срок годности: {StorageLife}ч\n" +

$"Тип: {Type}, вес: {Weight} кг\n" +

$"Стоимость: {Price}р., Количество на складе: {Quantity} шт.\n");

}

public void NonVirtualShow()

{

Console.WriteLine($"No virtual -> works as milks");

}

public override string ToString()

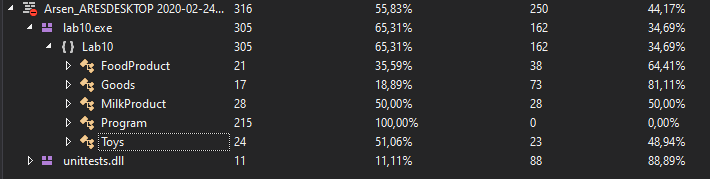
{

return $"Молочка: {Title}, {Manufacturer}, {Manufactured}, {StorageLife}ч.,\n{Type}, {Weight}, {Price}p., {Quantity}";

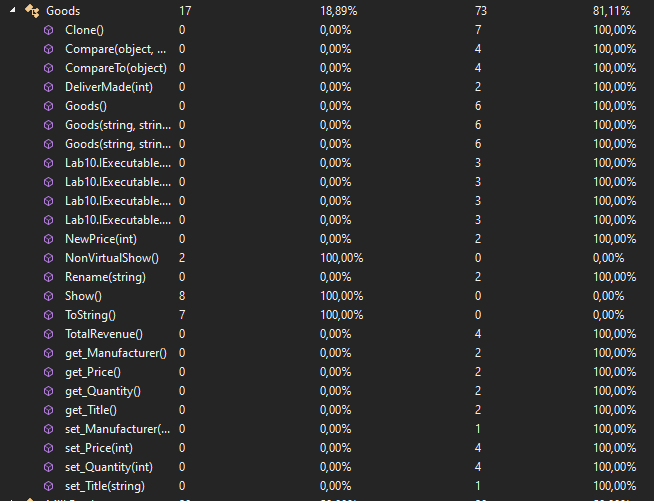
}

}

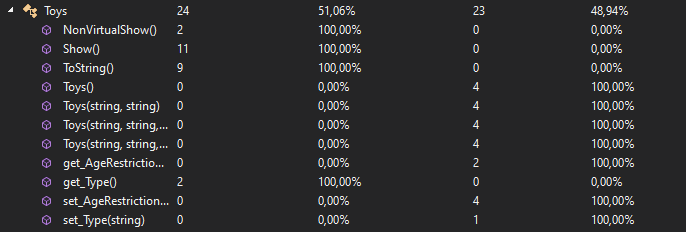
Тестирование

Общее покрытие:

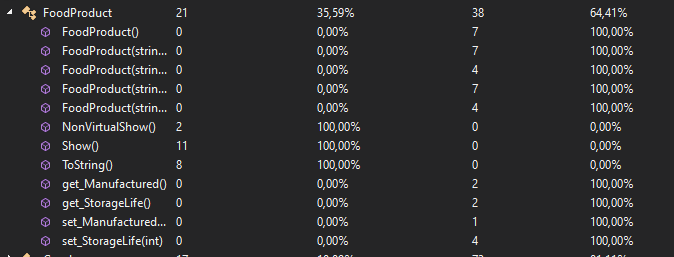
Class Goods:



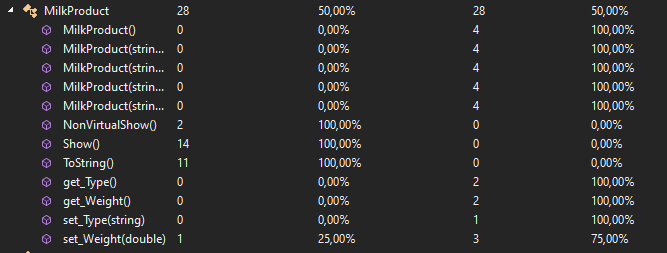
Class Toys:



Class FoodProduct:



Class MilkProduct



Листинг тестов:

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using Lab10;

namespace UnitTests

{

[TestClass]

public class UnitTest1

{

[TestMethod]

public void ComplexTestForConstrAndSort()

{

int expected = 0;

IExecutable[] goods = {

new Goods(),

new MilkProduct(),

new FoodProduct(),

new Toys(),

};

Array.Sort(goods);

Assert.AreEqual(expected, goods[3].TotalRevenue());

}

[TestMethod]

public void TestForContructors2()

{

int expected = 0;

IExecutable[] goods =

{

new Goods("Липтон", "Pepsi Co."),

new Goods("Кола", "Coca-Cola Co.", 99, 33),

new Toys("Ворона", "ВШЭ"),

new Toys("ЛЕГО Сити", "Lego", 6, "конструктор"),

new Toys("Монополия", "Hasbro", 800, 40, 3, "настольные игры"),

new FoodProduct("Колбаса", "Кунгурский"),

new FoodProduct("Йогурт", "Nestle", new DateTime(2020, 2, 25), 48),

new FoodProduct("Хлеб", "Хлебокомбинат", -50, -10),

new FoodProduct("Пирог", "Хлебокомбинат", 65, 40, new DateTime(), 72),

new MilkProduct("Простоквашино", "Danone"),

new MilkProduct("Домик в деревне", "Pepsi Co.", "молоко", 1.0),

new MilkProduct("Нытвенский", "Нытвенский", 33, 33, "Кефир", 1.0),

new MilkProduct("Домашний", "Ярмарка", 50, 90, new DateTime(2020, 2, 22), 72, "Творог", 0.5)

};

int actual = -1;

foreach(IExecutable good in goods)

{

if (good.Title() == "Хлеб" && good.Manufacturer() == "Хлебокомбинат") actual = good.Price() + good.Quantity();

}

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void CloneAndGoodsResetFunc1Test()

{

string expected = "Carlin";

Goods good = new Goods(expected, "F2");

Goods cloned = (Goods)good.Clone();

good.Rename("Hitech");

Assert.AreEqual(expected, cloned.Title);

}

[TestMethod]

public void CompareTest()

{

int expected = 0;

Goods good1 = new Goods("ENG", "RUS", 50, 90);

Goods good2 = new Goods("RUS", "ENG", 100, 45);

Assert.AreEqual(expected, good1.Compare(good1, good2));

}

[TestMethod]

public void GoodsResetFunc2Test()

{

int expected = 15 \* 30;

Goods good = new Goods("UFO", "MilkyWay", 100, 20);

good.NewPrice(15);

good.DeliverMade(30);

Assert.AreEqual(expected, good.TotalRevenue());

}

[TestMethod]

public void ToysTest()

{

bool expected = false;

Toys toy = new Toys("UFO", "WWA", 15, 15, -1, "IMF");

bool actual = toy.AgeRestriction > 0 && toy.Type == "IMF" && toy.Type != "";

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void FoodTest()

{

bool expected = true;

FoodProduct food = new FoodProduct("FOM", "UN", 10, 10, (new DateTime(2020, 2, 22)), 48);

FoodProduct food1 = new FoodProduct("FOM", "UN", 10, 10, (new DateTime(2020, 2, 20)), -118);

bool actual = food.Manufactured + (new TimeSpan(food.StorageLife, 0, 0)) > new DateTime(2020, 2, 23) && food1.StorageLife == 0;

Assert.AreEqual(expected, actual);

}

[TestMethod]

public void MilkTest()

{

bool expected = true;

MilkProduct milk = new MilkProduct("WWF", "CIS", 50, 1000, new DateTime(2020, 2, 22), 72, "молоко", 1.5);

MilkProduct milk1 = new MilkProduct("EU", "UNES", 1000, 10, new DateTime(2020, 2, 22), 60, "кефир", -1);

bool actual = (milk1.Type == "кефир" || milk1.Type == "молоко") && milk.Weight > 1.0;

Assert.AreEqual(expected, actual);

}

}

}