Задание за курсов проект на 11б обектно-ориентирано програмиране (ООП), с фокус върху C# (в средата на Visual Studio),

КРАЕН СРОК ЗА ПРЕДАВАНЕ 23.12.2026 г. или 06.01.2027 г.

Може по желание да създадете графичен интерфейс (напр. чрез ASP.NET или Windows Forms) във Visual Studio.

Всичко имате **40 теми** групирани логически в архитектурата за проекта,
12 уникални варианта (бизнес домейни) с еднаква сложност.

📑 ЗАДАНИЕ ЗА КУРСОВ ПРОЕКТ

Дисциплина: Обектно-ориентирано програмиране (ООП)
Среда за разработка: Microsoft Visual Studio (C# / .NET Console Application или Desktop App)

🎯 1. Обща цел на проекта

Да се разработи софтуерна система, прилагаща напреднали концепции за обектно-ориентирано проектиране, функционално програмиране и шаблони за дизайн. Системата трябва да моделира реални процеси в избран бизнес контекст и да поддържа въвеждане, филтриране, съхранение и обработка на данни чрез файлови потоци.

📂 2. Задължителни технически изисквания (Общи за всички варианти)

Всеки проект трябва задължително да интегрира следните конспектирани групи теми:

1. **Базова структура (Теми 03, 05, 07):** Всички бизнес обекти да притежават капсулирани полета, свойства (properties) с валидация и презаредени конструктори.
2. **Статични елементи (Теми 09, 11):** Да се реализира софтуерен брояч за обекти или конфигурационен клас със статични свойства и методи.
3. **Памет и ресурси (Теми 13, 15):** Демонстрирайте разбиране за Стек и Хийп чрез управление на референтни типове. Да се имплементира деструктор/финализатор (или `IDisposable`), който записва лог файл при унищожаване на обект.
4. **Шаблонни типове (Теми 17, 19, 21):** Създаване на собствен шаблонен (Generic) клас – хранилище за обекти (`Repository<T>`), съдържащ шаблонни методи с ограничители (Constraints) за референтен тип и наличие на конструктор (`where T : class, new()`).
5. **ООП Стълбове (Теми 23, 25, 27, 29, 31, 33, 37):**
   - Реализиране на **Наследяване** (базов абстрактен клас и поне два наследника).
   - Демонстриране на **Полиморфизъм** чрез презаписване (`override`) на виртуални/абстрактни методи и интерфейси.
6. **Алгоритми и функционално програмиране (Теми 39, 41, 43, 45, 47, 49, 51):**
   - Обектите да могат да се сортират по специфичен критерий (имплементиране на `IComparable` или `IComparer` / Компаратори).
   - Да се използват **Делегати**, **Ламбда изрази** и **LINQ заявки** за филтриране, сортиране и групиране на данните в колекциите.
7. **Събитийно програмиране (Теми 53, 55, 57, 59):** Имплементиране на събитийна комуникация. При настъпване на определено събитие (напр. критичен минимум на наличност, промяна на цена), софтуерни "слушатели" трябва автоматично да реагират чрез предаване на аргументи на събитието (`EventArgs`).
8. **Стабилност и Потоци (Теми 61, 63):** Пълно прихващане на грешки (`try-catch`) и хвърляне на потребителски изключения (`throw new CustomException`). Данните на системата да се записват и зареждат от текстови/бинарни файлове на диска.
9. **Шаблони за дизайн (Теми 65, 67, 69):** Вграждане на поне два шаблона по избор (напр. *Singleton* за мениджъра на системата, *Factory* за лесно създаване на обекти-наследници или *Observer* за събитията).

🎲 3. Варианти за учениците от 11б – ПГЕА (12 уникални теми)

Всеки ученик получава **един** от следните варианти по номер в клас, по който разработва гореописаната софтуерна архитектура:

№	Вариант	Име на системата	Базов Абстрактен Клас	Специфични Класове Наследници	Критично Събитие (Event)
13	Вариант1	**Автосалон (CarDealership)**	`Vehicle`	`Car`, `Motorcycle`, `Truck`	Продажба на автомобил над определена цена
14	Вариант2	**Хотелски резервации (HotelSystem)**	`Room`	`StandardRoom`, `Suite`, `Penthouse`	Резервация на стая в пиков сезон
15	Вариант3	**Онлайн магазин (E-Shop)**	`Product`	`Electronics`, `Clothing`, `Food`	Спадане на количеството на продукт под 5 бройки
16	Вариант4	**Университет (UniversitySystem)**	`User`	`Student`, `Professor`, `Staff`	Промяна на среден успех на студент
17	Вариант5	**Банкова система (BankSystem)**	`Account`	`SavingsAccount`, `CheckingAccount`	Теглене на сума, надвишаваща баланса
18	Вариант6	**Фитнес център (GymManager)**	`Membership`	`DailyPass`, `MonthlySubscription`	Изтичане на валидността на абонамент
19	Вариант7	**Библиотека (LibrarySystem)**	`LibraryItem`	`Book`, `Magazine`, `AudioBook`	Закъснение при връщане на книга
20	Вариант8	**Управление на ресторант (RestaurantApp)**	`MenuItem`	`Dish`, `Drink`, `Dessert`	Поръчка на алергенно ястие от клиент
21	Вариант9	**Автосервиз (AutoRepair)**	`ServiceJob`	`EngineRepair`, `BodyWork`, `Diagnostics`	Завършване на ремонт на автомобил
22	Вариант10	**Туристическа агенция (TravelAgency)**	`Trip`	`Excursion`, `Cruise`, `FlightPackage`	Запълване на максималния брой места за екскурзия
23	Вариант11	**Киносалон (CinemaSystem)**	`ShowEvent`	`MovieScreening`, `TheaterPlay`, `Concert`	Продажба на VIP билет
24	Вариант12	**Складово стопанство (Warehouse)**	`StorageItem`	`PerishableItem`, `FragileItem`	Надвишаване на максималния капацитет на склада

📋 4. Изисквания за предаване и защита

1. **Проектна папка:** Пълният сорс код на решението (`.sln` и `.cs` файлове), организиран по папки (домейни, шаблони, интерфейси).
2. **Файлова база данни:** Примерни текстови файлове, от които програмата автоматично чете данни при стартиране.
3. **Защита на проекта:** Всеки ученикът трябва да обясни разположението на обектите си в паметта (стек срещу хиип), как работи написаната LINQ заявка и да демонстрира прихващането на изключения в реално време.

примерен C# код

🎮 Вариант 13: Система за стрийминг на видеоигри (GameStreamingPlatform)

- **Базов абстрактен клас:** `StreamerContent`
- **Класове наследници:** `LiveStream`, `RecordedVideo`, `EsportsTournament`
- **Критично събитие (Event):** Преминаване на определен праг на зрители на живо (напр. над 10,000 зрители едновременно) или получаване на голямо софтуерно дарение (Donation).

🛠️ Примерна софтуерна архитектура за този вариант (название на темите):

• **Полета, свойства и конструктори (Теми 03, 05, 07):** Базовият клас съдържа капсулирана информация за заглавие на стрийма, име на играта, брой текущи зрители и променлива за приходи.

• **Статични елементи и памет (Теми 09, 11, 13, 15):** Статично свойство следи общия брой активни стриймъри в платформата в реално време. Имплементира се деструктор, който записва хронология (Log) в текстов файл, когато стриймът приключи и се изтрие от паметта.

• **Шаблонни класове и LINQ (Теми 17, 21, 47, 49):** Създава се шаблонен мениджър `PlatformHub<T> where T : StreamerContent`, който съхранява съдържанието и използва **LINQ** заявки, за да филтрира кои стриймове са с най-висок рейтинг или кои излъчват конкретна игра (напр. "Cyberpunk 2077").

• **Събития и делегати (Теми 53, 57, 59):** Когато броят на зрителите надвиши зададения праг, се задейства събитие `OnHypedStream`. Класът `ModeratorBot` (слушател) автоматично реагира, като изпраща съобщение в чата с аргументи, съдържащи точния брой зрители и бонус коефициент за стриймъра.

• **Изключения и потоци (Теми 61, 63):** Ако потребител се опита да влезе в стрийм с ограничение за възраст без да има навършени години, системата хвърля потребителско изключение `AgeRestrictionException`. При стартиране списъкът с игри се зарежда от външен текстов файл.

готов C# сорс код (скелет на проекта)

Вариант 14: Онлайн магазин с потребителска кошница (E-Shop with Shopping Cart).

Този вариант е изключително практичен за учениците, тъй като съчетава логиката на продуктовия каталог с динамичното управление на обекти в паметта (кошницата).

🛒 Вариант 14: Онлайн магазин с кошница

• **Базов абстрактен клас:** `Product`

• **Класове наследници:** `Electronics`, `Clothing`, `DigitalProduct` (напр. софтуерни лицензи или видеоигри)

• **Клас Кошница (Shopping Cart):** `ShoppingCart<T> where T : Product` — шаблонен клас, който управлява селекцията на текущия потребител.

• **Критично събитие (Event):** Финализиране на поръчка (Checkout) или спадане на наличността на продукт в склада под критичен минимум.

🛠️ Архитектура и интеграция на темите в този вариант:

• **Полета, свойства и конструктори (Теми 03, 05, 07):** Всеки `Product` има капсулирани свойства (Име, Цена, Наличност в склада). Чрез `set` селекторите се прави валидация — цената и количеството не могат да бъдат отрицателни числа.

• **Статични елементи и памет (Теми 09, 11, 13, 15):** Статично свойство в магазина пази общия оборот от всички продажби или текущия ДДС процент. При изтриване на празна кошница от паметта, деструкторът записва лог за прекратената сесия.

• **Шаблонна Кошница и LINQ (Теми 17, 21, 47, 49):** Класът `ShoppingCart<T>` съдържа списък от продукти. Използват се **LINQ** функции (напр. `.Sum(p => p.Price)`, `.Where()`, `.GroupBy()`) за автоматично изчисляване на общата стойност, филтриране на дигитални продукти и групиране по категории.

• **ООП Стълбове и Интерфейси (Теми 27, 29, 31, 37):** Продуктите имплементират интерфейс `IDiscountachable`. Методът `CalculateFinalPrice()` се презаписва (`override`), тъй като дрехите могат да имат сезонно намаление в проценти, електрониката — фиксирана отстъпка, а дигиталните продукти — липса на такса за доставка.

• **Събития и комуникация (Теми 53, 55, 57):** Когато потребителят натисне "Плащане", кошницата задейства събитието `OnOrderPlaced`. Слушателят `InventoryManager` автоматично намалява количествата в склада с аргументи от събитието (`OrderEventArgs`), съдържащи списъка с купени стоки.

• **Изключения и потоци (Теми 61, 63):** Ако потребителят се опита да добави в кошницата продукт, чието количество в склада е 0, системата хвърля изключение `OutOfStockException`. При стартиране каталогът от продукти се зарежда от файл `products.txt`, а при покупка се генерира текстов файл `invoice.txt` (фактура).

готов и работещ C# код за онлайн магазин с кошница:

.

Зареждане и дефинирането на базата с продукти:
Шаблонните класове (Generic Repository),
Полиморфизмът
Работата с файлови потоци (Теми 17, 29, 63).

В обектно-ориентираното програмиране с C# базата от данни се дефинира по два начина, които работят в синхрон: **софтуерна структура (класове)** и **текстов файл (products.txt)**, който симулира реална база данни.

---

1. Текстовият файл (`products.txt`)

За да може програмата да чете продуктите динамично, те се записват във файл, където първата дума на всеки ред указва типа на продукта (за полиморфно създаване), а останалите данни са разделени с разделител (напр. запетая или вертикална черта `|`):

text

```
Electronics | Лаптоп ASUS|1200.00|5|24
Clothing | Яке Nike|150.00|10|L
Drink | Кафе Лате|4.50|50|0.350
Electronics | Смартфон Samsung|800.00|3|12
Clothing | Дънки Levi's|110.00|8|M
```

Забележка: Последният параметър е специфичен за типа продукт (Електроника -> Гаранция в месеци; Дрехи -> Размер; Напитки -> Обем в литри).

---

2. Как се дефинира това в C# кода?

Ето как изглежда архитектурната дефиниция на класовете и базовия метод за зареждане. Този код показва как от един текстов файл, чрез **Полиморфизъм**, се създават правилните обекти в паметта.

csharp

```
using System;
using System.Collections.Generic;
using System.IO;

namespace EShopSystem
{
    // --- ТЕМА 27: АБСТРАКЦИЯ И ИНТЕРФЕЙСИ ---
    public interface IDiscountable
    {
        decimal CalculateFinalPrice();
    }

    // --- ТЕМА 33: АБСТРАКТНИ КЛАСОВЕ И ТЕМА 05: ПОЛЕТА И СВОЙСТВА ---
    public abstract class Product : IDiscountable
    {
        private string name;
        private decimal price;
        private int quantity;

        public string Name
        {
            get => name;
            set => name = !string.IsNullOrWhiteSpace(value) ? value : throw new
ArgumentException("Името не може да е празно.");
        }

        public decimal Price
```

```
            {
                get => price;
                set => price = value >= 0 ? value : throw new
ArgumentException("Цената не може да е отрицателна.");
            }

            public int Quantity
            {
                get => quantity;
                set => quantity = value >= 0 ? value : throw new
ArgumentException("Количеството не може да е отрицателно.");
            }

            // ТЕМА 07: МЕТОДИ И КОНСТРУКТОРИ
            protected Product(string name, decimal price, int quantity)
            {
                Name = name;
                Price = price;
                Quantity = quantity;
            }

            // ТЕМА 31: ПРЕЗАПИСВАНЕ (OVERRIDE)
            public abstract decimal CalculateFinalPrice();

            public override string ToString()
            {
                return $"{Name} - Цена: {CalculateFinalPrice():F2} лв. (Наличност:
{Quantity} бр.)";
            }
        }

        // --- ТЕМА 23: НАСЛЕДЯВАНЕ (КОНКРЕТНИ ПРОДУКТИ) ---
        public class Electronics : Product
        {
            public int WarrantyMonths { get; set; }

            public Electronics(string name, decimal price, int quantity, int
warrantyMonths)
                : base(name, price, quantity)
            {
                WarrantyMonths = warrantyMonths;
            }

            public override decimal CalculateFinalPrice() => Price; // Електрониката
няма автоматична отстъпка
        }

        public class Clothing : Product
        {
            public string Size { get; set; }

            public Clothing(string name, decimal price, int quantity, string size)
                : base(name, price, quantity)
            {
                Size = size;
            }

            public override decimal CalculateFinalPrice() => Price * 0.90m; // 10%
сезонно намаление за дрехи
        }

        public class Drink : Product
        {
            public double VolumeLiters { get; set; }
```

```
            public Drink(string name, decimal price, int quantity, double
volumeLiters)
                : base(name, price, quantity)
            {
                VolumeLiters = volumeLiters;
            }

            public override decimal CalculateFinalPrice() => Price;
        }

        // --- ТЕМА 17 & 21: ШАБЛОНЕН КЛАС (REPOSITORY) ЗА БАЗАТА С ПРОДУКТИ ---
        public class ProductRepository<T> where T : Product
        {
            private List<T> products = new List<T>();

            public void Add(T product) => products.Add(product);
            public List<T> GetAll() => products;

            // ТЕМА 63: РАБОТА С ПОТОЦИ И ФАЙЛОВЕ
            // Фабричен метод за полиморфно зареждане на "Базата данни"
            public static List<Product> LoadDatabaseFromFile(string filePath)
            {
                List<Product> loadedProducts = new List<Product>();

                if (!File.Exists(filePath))
                {
                    throw new FileNotFoundException($"Файлът {filePath} не е
намерен! Моля, създайте го.");
                }

                // Четем файла ред по ред
                string[] lines = File.ReadAllLines(filePath);
                foreach (var line in lines)
                {
                    if (string.IsNullOrWhiteSpace(line)) continue;

                    string[] parts = line.Split('|');
                    string type = parts[0];
                    string name = parts[1];
                    decimal price = decimal.Parse(parts[2]);
                    int quantity = int.Parse(parts[3]);

                    // ТЕМА 29: ПОЛИМОРФИЗЪМ ПРИ СЪЗДАВАНЕ НА ОБЕКТИТЕ
                    switch (type)
                    {
                        case "Electronics":
                            int warranty = int.Parse(parts[4]);
                            loadedProducts.Add(new Electronics(name, price,
quantity, warranty));
                            break;
                        case "Clothing":
                            string size = parts[4];
                            loadedProducts.Add(new Clothing(name, price, quantity,
size));
                            break;
                        case "Drink":
                            double volume = double.Parse(parts[4]);
                            loadedProducts.Add(new Drink(name, price, quantity,
volume));
                            break;
                    }
                }
                return loadedProducts;
            }
        }
    }
```

Описание на структурата на кода?

1. **Ясно разделение:** Виждат се как данните от текстовия файл се превръщат в истински C# обекти чрез `switch` логиката.
2. **Лесно разширение:** Ако утре искате да добавите нова категория (напр. *Книги* или *ИГРАЧКИ*), просто трябва да създаде нов клас-наследник и да добави един ред в `switch` структурата.

останалата част от проекта — същинския клас за
Кошницата (ShoppingCart) със събитието за покупка и
LINQ заявките в главната програма?

Работещ сорс код за **Онлайн магазин с потребителска кошница** (включващ количка, LINQ заявки, потребителски събития и обработка на грешки). Програмата демонстрира нагледно как се обвързват всички сложни теми от конспекта.

Можете директно да копирате и този код, като **скелет за курсовия проект** (Skeleton Project).

🛒 ПЪЛЕН СОРС КОД (Program.cs)

csharp

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;

namespace EShopSystem
{
    #region 1. СЪБИТИЯ И АРГУМЕНТИ (Теми 53, 55, 57)

    // Тема 57: Комуникация между обекти. Аргументи на събития
    public class OrderEventArgs : EventArgs
    {
        public List<Product> OrderedItems { get; private set; }
        public decimal TotalPrice { get; private set; }

        public OrderEventArgs(List<Product> items, decimal total)
        {
            OrderedItems = items;
            TotalPrice = total;
        }
    }

    #endregion

    #region 2. ИЗКЛЮЧЕНИЯ (Тема 61)

    // Тема 61: Прихващане и хвърляне на изключения (Потребителско изключение)
    public class OutOfStockException : Exception
    {
        public OutOfStockException(string message) : base(message) { }
    }

    #endregion

    #region 3. ИНТЕРФЕЙСИ И БАЗОВИ КЛАСОВЕ (Теми 03, 05, 07, 23, 27, 31, 33)

    // Тема 27 & 37: Абстракция. Полиморфизъм чрез интерфейси
    public interface IDiscountable
    {
        decimal CalculateFinalPrice();
    }

    // Тема 33: Абстрактни класове и полиморфизъм
```

```
        // Тема 03 & 05: Дефиниране на класове, Полета и свойства
        public abstract class Product : IDiscountable, IComparable<Product>
        {
            private string name;
            private decimal price;
            private int quantity;

            public string Name
            {
                get => name;
                set => name = !string.IsNullOrWhiteSpace(value) ? value : throw new
ArgumentException("Името не може да бъде празно.");
            }

            public decimal Price
            {
                get => price;
                set => price = value >= 0 ? value : throw new
ArgumentException("Цената не може да бъде отрицателна.");
            }

            public int Quantity
            {
                get => quantity;
                set => quantity = value >= 0 ? value : throw new
ArgumentException("Количеството не може да бъде отрицателно.");
            }

            // Тема 07: Методи и конструктори
            protected Product(string name, decimal price, int quantity)
            {
                Name = name;
                Price = price;
                Quantity = quantity;
            }

            // Тема 31: Презаписване (override) на методи
            public abstract decimal CalculateFinalPrice();

            // Тема 41: Итератори. Компаратори (Сортиране по цена по подразбиране)
            public int CompareTo(Product other)
            {
                if (other == null) return 1;
                return
this.CalculateFinalPrice().CompareTo(other.CalculateFinalPrice());
            }

            public override string ToString()
            {
                return $"[Продукт] {Name} | Каталожна цена: {Price:F2} лв. | Крайна
цена: {CalculateFinalPrice():F2} лв.";
            }
        }

        // Тема 23 & 25: Наследяване и преизползване на класовете
        public class Electronics : Product
        {
            public int WarrantyMonths { get; set; }

            public Electronics(string name, decimal price, int quantity, int
warrantyMonths)
                : base(name, price, quantity)
            {
                WarrantyMonths = warrantyMonths;
            }
```

```
            // Електрониката няма отстъпка по подразбиране
            public override decimal CalculateFinalPrice() => Price;

            public override string ToString() => base.ToString() + $" (Гаранция:
{WarrantyMonths} мес.)";
        }

        public class Clothing : Product
        {
            public string Size { get; set; }

            public Clothing(string name, decimal price, int quantity, string size)
                : base(name, price, quantity)
            {
                Size = size;
            }

            // Тема 29: Демонстрация на Полиморфизъм - Дрехите имат 10% сезонно
намаление
            public override decimal CalculateFinalPrice() => Price * 0.90m;

            public override string ToString() => base.ToString() + $" (Размер:
{Size}, Включена 10% отстъпка)";
        }

        public class Drink : Product
        {
            public double VolumeLiters { get; set; }

            public Drink(string name, decimal price, int quantity, double
volumeLiters)
                : base(name, price, quantity)
            {
                VolumeLiters = volumeLiters;
            }

            public override decimal CalculateFinalPrice() => Price;

            public override string ToString() => base.ToString() + $" (Обем:
{VolumeLiters}л)";
        }

        #endregion

        #region 4. ШАБЛОНЕН КЛАС И КОШНИЦА (Теми 11, 15, 17, 21, 51, 55, 59)

        // Тема 17 & 21: Шаблонни класове и ограничители (where T : Product)
        public class ShoppingCart<T> where T : Product
        {
            // Тема 13: Памет, стек, хиип. Списъкът се заделя динамично в Хийпа.
            private List<T> items = new List<T>();

            // Тема 11: Статични полета и свойства (Глобален брояч на създадени
кошници в системата)
            public static int TotalCartsCreated { get; private set; }

            // Тема 53 & 55: Комуникация между обекти. Делегати и Събития.
            public event EventHandler<OrderEventArgs> OnCheckout;

            // Тема 09: Статичен конструктор
            static ShoppingCart()
            {
                TotalCartsCreated = 0;
            }

            public ShoppingCart()
```

```
            {
                TotalCartsCreated++;
            }

            // Тема 15: Финализатори и деструктори
            ~ShoppingCart()
            {
                // Извиква се, когато GC изчисти обекта от паметта (логване на
събитието)
                Console.WriteLine("\n[Системен Лог] Обектът 'Кошница' е премахнат от
паметта.");
            }

            public void AddItem(T item)
            {
                // Тема 61: Хвърляне на изключение при липса на наличност в склада
                if (item.Quantity <= 0)
                {
                    throw new OutOfStockException($"Продуктът '{item.Name}' е
изчерпан в склада и не може да бъде добавен!");
                }

                items.Add(item);
                item.Quantity--; // Намаляваме наличността временно в реално време
                Console.WriteLine($"Успешно добавихте '{item.Name}' в кошницата.");
            }

            // Тема 47 & 49: Използване на LINQ за изчисление на общата сума
            public decimal GetTotal()
            {
                return items.Sum(x => x.CalculateFinalPrice());
            }

            public List<T> GetItems() => items;

            public void Checkout()
            {
                if (items.Count == 0)
                {
                    Console.WriteLine("Кошницата е празна. Няма какво да
финализирате.");
                    return;
                }

                decimal total = GetTotal();
                Console.WriteLine($"\n--- Финализиране на поръчката на стойност
{total:F2} лв. ---");

                // Задействане на събитието (ако има регистрирани слушатели)
                OnCheckout?.Invoke(this, new OrderEventArgs(new
List<Product>(items), total));

                items.Clear(); // Изпразваме кошницата след покупка
            }
        }

        // Тема 59: Слушател за събитие (EventHandler)
        public class InventoryManager
        {
            // При финализиране на поръчка, този метод автоматично реагира и записва
касова бележка
            public void OnOrderPlaced(object sender, OrderEventArgs e)
            {
                Console.WriteLine("[Слушател (InventoryManager)]: Системата обработи
плащането успешно.");
```

```
                // Тема 63: Работа с потоци и файлове. Записване на фактура на
диска.
                string invoicePath = "invoice.txt";
                using (StreamWriter writer = new StreamWriter(invoicePath))
                {
                    writer.WriteLine($"======= КАСОВА БЕЛЕЖКА / ФАКТУРА =======");
                    writer.WriteLine($"Дата: {DateTime.Now}");
                    writer.WriteLine("Закупени стоки:");
                    foreach (var item in e.OrderedItems)
                    {
                        writer.WriteLine($"- {item.Name}
({item.CalculateFinalPrice():F2} лв.)");
                    }
                    writer.WriteLine($"---------------------------------------");
                    writer.WriteLine($"ОБЩО ПЛАТЕНО: {e.TotalPrice:F2} лв.");
                    writer.WriteLine($"=======================================");
                }
                Console.WriteLine($"[Система]: Генериран е текстов файл с фактура:
'{Path.GetFullPath(invoicePath)}'");
            }
        }

        #endregion

        #region 5. ГЛАВНА ПРОГРАМА (Теми 39, 45, 47, 49, 63)

        class Program
        {
            static void Main(string[] args)
            {
                // Подсигуряване на правилно четене на символи на български в
конзолата
                Console.OutputEncoding = System.Text.Encoding.UTF8;
                Console.InputEncoding = System.Text.Encoding.UTF8;

                // 1. Създаване на примерен файл с база от данни на диска (Тема 63)
                string dbPath = "products.txt";
                CreateMockDatabaseFile(dbPath);

                // 2. Инициализиране на списък с продукти (База данни в паметта)
                List<Product> database = new List<Product>();

                try
                {
                    // Тема 63 & 29: Четем от файл и чрез полиморфизъм пълним
списъка
                    database = LoadDatabaseFromFile(dbPath);
                }
                catch (Exception ex)
                {
                    Console.WriteLine($"Грешка при зареждане на базата данни:
{ex.Message}");
                    return;
                }

                Console.WriteLine("======= ДОСТЪПНИ ПРОДУКТИ В МАГАЗИНА =======");
                database.ForEach(p => Console.WriteLine(p));

                // 3. Създаване на Кошница и Абониране на Слушател (Тема 55 & 59)
                ShoppingCart<Product> myCart = new ShoppingCart<Product>();
                InventoryManager inventoryManager = new InventoryManager();

                // Абониране за събитието Checkout
                myCart.OnCheckout += inventoryManager.OnOrderPlaced;
```

```
                Console.WriteLine($"\n[Системна информация] Създадени кошници до
момента: {ShoppingCart<Product>.TotalCartsCreated}");

                // 4. Симулация на пазаруване с прихващане на грешки (Тема 61)
                Console.WriteLine("\n======= СИМУЛАЦИЯ НА ПАЗАРУВАНЕ =======");
                try
                {
```

```
    // Успешно добавяне (Има ги в наличност)
myCart.AddItem(database[0]); // Лаптоп ASUS
myCart.AddItem(database[1]); // Яке Nike
    // Опит за добавяне на изчерпан продукт (Кафе Лате е с количество 0 във
файла)
myCart.AddItem(database[2]);
}
catch (OutOfStockException ex)
{
// Тема 61: Хващане на нашето специфично изключение
Console.ForegroundColor = ConsoleColor.Red;
Console.WriteLine($"[ГРЕШКА ПРИ ПАЗАРУВАНЕ]: {ex.Message}");
Console.ResetColor();
}
catch (Exception ex)
{
Console.WriteLine($"Обща грешка: {ex.Message}");
}
    // 5. ТЕМИ 47 & 49: ЕЛЕМЕНТИ ОТ ФУНКЦИОНАЛНОТО ПРОГРАМИРАНЕ И LINQ
Console.WriteLine("\n======= ФУНКЦИОНАЛНО ПРОГРАМИРАНЕ И LINQ (Анализ на
количката) =======");
    // Филтриране чрез Ламбда израз: Намиране на продукти в количката на цена над
100 лв.
var expensiveItems = myCart.GetItems().Where(p => p.CalculateFinalPrice() >
100).ToList();
Console.WriteLine($"Продукти в количката с цена над 100 лв.:
{expensiveItems.Count}");
expensiveItems.ForEach(p => Console.WriteLine($" - {p.Name}"));
    // Сортиране на останалите продукти в магазина чрез LINQ възходящо по цена
Console.WriteLine("\nВсички продукти от каталога, сортирани по цена (чрез
LINQ):");
var sortedCatalog = database.OrderBy(p => p.CalculateFinalPrice());
foreach (var item in sortedCatalog)
{
Console.WriteLine($" * {item.Name} -> {item.CalculateFinalPrice():F2} лв.");
}
    // 6. Финализиране на покупката (Задейства събитието и записва файла)
myCart.Checkout();
    Console.WriteLine("\nНатиснете [Enter] за край на програмата...");
Console.ReadLine();
}
    #region Помощни методи за работа с файлове (Тема 63)
    private static void CreateMockDatabaseFile(string path)
{
// Създаваме примерен тестов файл
// Формат: Тип|Име|КаталожнаЦена|Налично Количество|СпецифичноСвойство
string[] mockData = new string[]
{
"Electronics|Лаптоп ASUS|1200.00|5|24",
"Clothing|Яке Nike|150.00|10|L",
"Drink|Кафе Лате (ИЗЧЕРПАНО)|4.50|0|0.350", // Количеството е 0 за тест на
изключението
"Electronics|Смартфон Samsung|800.00|3|12"
};
```

```
File.WriteAllLines(path, mockData);
}
    private static List LoadDatabaseFromFile(string filePath)
{
List products = new List();
    if (!File.Exists(filePath)) return products;
    string[] lines = File.ReadAllLines(filePath);
foreach (var line in lines)
{
if (string.IsNullOrWhiteSpace(line)) continue;
    string[] parts = line.Split('|');
string type = parts[0];
string name = parts[1];
decimal price = decimal.Parse(parts[2]);
int quantity = int.Parse(parts[3]);
    // Тема 29: Полиморфизъм при запълване на базата данни
switch (type)
{
case "Electronics":
int warranty = int.Parse(parts[4]);
products.Add(new Electronics(name, price, quantity, warranty));
break;
case "Clothing":
string size = parts[4];
products.Add(new Clothing(name, price, quantity, size));
break;
case "Drink":
double volume = double.Parse(parts[4]);
products.Add(new Drink(name, price, quantity, volume));
break;
}
}
return products;
}
    #endregion
}
    #endregion
}
```

    ### 💡 Какво показва този код?

***Капсулация:** Properties (`Name`, `Price`) правят активна валидация при зареждане на данните.

***Полиморфизъм:** Методът `CalculateFinalPrice()` връща пълната цена за техниката, но автоматично приспада 10% за дрехите.

***Събитийна архитектура:** Класът `ShoppingCart` изобщо не знае как се печата фактура. Той просто задейства събитието `OnCheckout`, а външният слушател `InventoryManager` улавя събитието и записва файла `invoice.txt`.

Примерен документ с подробни **Критерии за оценяване на курсовия проект**.
И как да бъде оформен.

📊 КРИТЕРИИ ЗА ОЦЕНЯВАНЕ НА КУРСОВИЯ ПРОЕКТ

Максималният брой точки, който софтуерната система може да събере, е **100 точки**. Те са разпределени в 4 основни компонента:

---

1. Архитектура на кода и ООП концепции (Макс. 30 точки)

- **Капсулация и свойства (Теми 03, 05, 07):** Всички полета са защитени (`private`/`protected`). Свойствата (`properties`) притежават работеща бизнес валидация (напр. цената или количеството не могат да бъдат отрицателни). (**7 точки**)
- **Наследяване и Абстракция (Теми 23, 27, 33):** Налице е базов абстрактен клас/интерфейс и поне три смислени класа наследници. Базовият клас не може да се инстанцира директно. (**8 точки**)
- **Полиморфизъм (Теми 29, 31, 37):** Методът за изчисляване на цена (или специфично действие) е абстрактен/виртуален и е коректно презаписан (`override`) в наследниците с различна бизнес логика. (**8 точки**)
- **Статични елементи (Теми 09, 11):** Наличие на работещ статичен брояч (напр. за брой създадени обекти/кошници) или статичен конфигурационен метод. (**7 точки**)

2. Шаблонни типове и Функционално програмиране (Макс. 25 точки)

- **Шаблонни класове (Теми 17, 19, 21):** Количката или хранилището за данни е реализирано като Generic клас с рестрикция (`where T : Product` или сходна), предпазваща от подаване на грешни типове данни. (**10 точки**)
- **Ламбда изрази и LINQ (Теми 45, 47, 49):** Наличие на поне две LINQ заявки (напр. филтриране `.Where()`, сортиране `.OrderBy()`, сумиране `.Sum()`), които обработват колекциите по модерен и оптимизиран начин. (**10 точки**)
- **Компаратори (Тема 41):** Класовете имплементират интерфейса `IComparable` или външен компоратор за автоматично сортиране на обектите по определен признак. (**5 точки**)

3. Събития, Потоци и Изключения (Макс. 25 точки)

- **Събитийно програмиране (Теми 53, 55, 57, 59):** Имплементирана е слаба връзка (Loose Coupling) чрез събития (`event`). Има създаден издател на събитието (Publisher), подават се специфични аргументи (`EventArgs`) и външен слушател (Subscriber) реагира автоматично. (**10 точки**)
- **Работа с файлове (Тема 63):** Програмата успешно прочита базата от данни от текстов файл при стартиране и записва изходен документ (напр. фактура/касова бележка) на диска при финализиране. (**10 точки**)
- **Управление на изключения (Тема 61):** Програмата не "гърми" при некоректни данни. Хвърлят се потребителски изключения (`throw new ...`) и се прихващат безопасно в `try-catch` блокове. (**5 точки**)

4. Документация, Защита и Чист код (Макс. 20 точки)

- **Чист код (Clean Code):** Правилно именуване на променливи, класове и методи (PascalCase и camelCase). Липса на излишен и повтарящ се код. Кодът е подреден и лесен за четене. (**5 точки**)
- **Разбиране на паметта (Тема 13, 15):** По време на защитата ученикът може да обясни разликата между Стек и Хийп, къде отиват стойностните и къде референтните типове и каква е ролята на деструктора/Garbage Collector-а. (**10 точки**)
- **Работа със средите (Visual Studio):** Ученикът демонстрира свободно боравене със средата, умее да стартира проекта и да покаже къде се генерират текстовите файлове. (**5 точки**)

📈 Актуализирана скала за превръщане на точките в оценка:

- **От 00 до 29 т.** — Слаб (2)
- **От 30 до 46 т.** — Среден (3)
- **От 47 до 64 т.** — Добър (4)
- **От 65 до 82 т.** — Мн. Добър (5)
- **От 83 до 100т.** — Отличен (6)