

ДЪРЖАВЕН ЗРЕЛОСТЕН ИЗПИТ ПО ИНФОРМАТИКА

24 август 2026 г.
ПРОФИЛИРАНА ПОДГОТОВКА
ВАРИАНТ 2

Ключ с верните отговори

Въпрос №	Верен отговор	Брой точки
1.	В	1
2.	А	1
3.	В	1
4.	Б	1
5.	Г	1
6.	В	1
7.	Б	1
8.	В	1
9.	Б	1
10.	Б	1
11.	А	1
12.	Б	1
13.	А	1
14.	Г	1
15.	Г	1
16.	А	1

Задача 17. – 2 точки

4: FOREST

7: STONE

Задача 18. – 2 точки

Client и Person

ElectricalCompany и Client

Задача 19. – 2 точки

(1) protected double batteryLevel

(2) public bool UpdateSoftware(int version) (за C#)

Public Boolean UpdateSoftware(int version) (за Java)

Задача 20. – 3 точки

- (1) Invalid age!
Validation sequence ended.
- (2) Input is not a integer number!
Validation sequence ended.
- (3) Age is valid.
Validation sequence ended.

Задача 21. – 3 точки

7

Black Eagle

Eagle

Задача 22. – 3 точки

- (1) d.dept_id = e.dept_id
- (2) d.dept_name
- (3) average_salary или AVG(e.salary)

Задача 23. – 3 точки

- (1) IndexOf или BinarySearch (за C#) / binarySearch (за Java)
- (2) index >= 0
- (3) Намерен на позиция: 5

Задача 24. – 3 точки

	За C#	За Java
(1)	Abs	abs
(2)	Floor	floor
(3)	Ceiling	ceil

Задача 25. – 3 точки

```
1 if (a > 0 && b > 0 && c > 0)
2 if (a + b > c && a + c > b && b + c > a){
3 double p = (a + b + c) / 2;
```

Задача 26. – 15 точки

Примерно решение:

За C#

```
namespace DZI2026
{
    internal class Task26
    {
        static void Main()
        {
```

```

        //въвеждат се броя продажби по месеци
        int[] sales =
Console.ReadLine().Split().Select(int.Parse).ToArray();
        int bonusValue = int.Parse(Console.ReadLine()); //сумата
на еднократен бонус
        int sanctionValue = int.Parse(Console.ReadLine()); //
сумата на еднократна санкция

        int maxSale = sales[0]; //текуща максимална продажба
        int minSale = sales[0]; //текуща минимална продажба

        int bonusCount = 0; //брояч - бонуси
        int sanctionCount = 0; //брояч - санкции

        for (int i = 1; i < 12; i++)
        {
            if (sales[i] > maxSale)
            {
                maxSale = sales[i];
                bonusCount++;
            }
            else if (sales[i] < minSale)
            {
                minSale = sales[i];
                sanctionCount++;
            }
        }

        int finalSum = bonusCount * bonusValue - sanctionCount *
sanctionValue;

        Console.WriteLine($"{bonusCount} {sanctionCount}");
        Console.WriteLine(finalSum);
    }
}

```

3a Java

```

package DZI2026;
import java.util.Scanner;

public class Task26 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int[] sales = new int[12];
        //въвеждат се броя продажби по месеци
        for (int i = 0; i < 12; i++) {
            sales[i] = sc.nextInt();
        }
    }
}

```

```

        int bonusValue = sc.nextInt(); //сумата на еднократен бонус
        int sanctionValue = sc.nextInt(); //сумата на еднократна
санкция

        int maxSale = sales[0]; //текуща максимална продажба
        int minSale = sales[0]; //текуща минимална продажба

        int bonusCount = 0; //брояч - бонуси
        int sanctionCount = 0; //брояч - санкции

        for (int i = 1; i < 12; i++) {
            if (sales[i] > maxSale) {
                maxSale = sales[i];
                bonusCount++;
            } else if (sales[i] < minSale) {
                minSale = sales[i];
                sanctionCount++;
            }
        }

        int finalSum = bonusCount * bonusValue - sanctionCount *
санctionValue;

        System.out.println(bonusCount + " " + sanctionCount);
        System.out.println(finalSum);
        sc.close();
    }
}

```

Задача 27. – 20 точки

Примерно решение

-- Създаване на база данни

```
CREATE DATABASE BookStore;
```

```
USE BookStore;
```

-- А)

```
CREATE TABLE Publishers (
    publisher_id INT IDENTITY PRIMARY KEY,
    name VARCHAR(50) NOT NULL
);
```

-- Б)

```
CREATE TABLE Books (
    book_id INT IDENTITY PRIMARY KEY,
    title VARCHAR(50) NOT NULL,
    author VARCHAR (50) NOT NULL,
    year INT NOT NULL,
    quantity INT NOT NULL,
    price DECIMAL(8,2) NOT NULL,
```

```

        publisher_id INT NOT NULL,
        FOREIGN KEY (publisher_id) REFERENCES Publishers(publisher_id)
    );

-- B)
INSERT INTO Publishers (name) VALUES
('Penguin Books'), ('HarperCollins'), ('Oxford Press'),
('Springer'), ('Pearson');

-- Г)
INSERT INTO Books (title, author, YEAR, quantity, price, publisher_id)
VALUES ('Database Systems', 'Thomas Connolly', 2018, 12, 24.9, 5),
('Learning SQL', 'Alan Beaulieu', 2022, 8, 19.99, 1),
('Advanced Java', 'Herbert Schildt', 2017, 5, 25, 3),
('Web Development', 'Jon Duckett', 2019, 11, 12.5, 1),
('Algorithms', 'Robert Sedgewick', 2015, 6, 20, 3),
('Spring Framework', 'Craig Walls', 2022, 9, 29.99, 2),
('Artificial Intelligence', 'Stuart Russell', 2021, 0, 15, 3);

-- Д)
UPDATE Books
SET quantity = quantity - 1
WHERE book_id = 2;

-- E)
SELECT title, author, year, quantity, price
FROM Books
WHERE year > 2020
ORDER BY title;

-- Ж)
SELECT title, author, quantity, price
FROM Books AS b
JOIN Publishers AS p ON b.publisher_id = p.publisher_id
WHERE name = 'Oxford Press' AND quantity > 0

-- З)
SELECT title, name, quantity, price, quantity*price AS total
FROM Books AS b
JOIN Publishers AS p ON b.publisher_id = p.publisher_id
WHERE quantity < 10;

-- И)
SELECT name, SUM(quantity) AS 'total books'
FROM Publishers AS p
LEFT JOIN Books AS b ON p.publisher_id = b.publisher_id
GROUP BY name
ORDER BY SUM(quantity) DESC -- или ORDER BY 'total books' DESC

```

Задача 28. – 25 точки

Примерно решение

За C#

namespace DZI2026

```
{
    internal abstract class Product
    {
        private string barcode;
        private string name;
        private int quantity;
        private double price;
        public string Barcode
        {
            get { return barcode; }
            set
            {
                if (value.Length != 10)
                {
                    throw new Exception("Invalid barcode!");
                }
                barcode = value;
            }
        }
        public string Name
        {
            get { return name; }
            set { name = value; }
        }
        public int Quantity
        {
            get { return quantity; }
            set
            {
                if (value < 0)
                {
                    throw new Exception("Invalid quantity!");
                }
                quantity = value;
            }
        }
        public double Price
        {
            get { return price; }
            set { price = value; }
        }

        public Product(string barcode, string name, int quantity,
double price)
        {

```

```

        Barcode = barcode;
        Name = name;
        Quantity = quantity;
        Price = price;
    }
    public override string ToString()
    {
        return $"{Barcode} {Name} - {Quantity}; {Price:F2} eur";
    }
}
internal class Drink : Product
{
    private int volume;
    public int Volume
    {
        get { return volume; }
        set { volume = value; }
    }
    public Drink(string barcode, string name, int quantity, double
price, int volume)
        : base(barcode, name, quantity, price)
    {
        Volume = volume;
    }
    public override string ToString()
    {
        return base.ToString() + $"; {Volume} ml.";
    }
}
internal class Snack : Product
{
    private int weight;
    public int Weight
    {
        get { return weight; }
        set { weight = value; }
    }
    public Snack(string barcode, string name, int quantity, double
price, int weight) : base(barcode, name, quantity, price)
    {
        Weight = weight;
    }
    public override string ToString()
    {
        return base.ToString() + $"; {Weight} gr.";
    }
}
internal class VendingMachine
{

```

```

List<Product> products;
public VendingMachine()
{
    products = new List<Product>();
}
public void AddProduct(Product product)
{
    foreach (Product p in products)
    {
        if (p.Barcode == product.Barcode)
        {
            throw new Exception("Product with this code
already exists.");
        }
    }
    products.Add(product);
}
public void BuyProduct(string code)
{
    Product product = null;
    foreach (Product p in products)
    {
        if (p.Barcode == code)
        {
            product = p;
            break;
        }
    }
    if (product.Quantity == 0)
    {
        throw new Exception("Product is out of stock.");
    }
    product.Quantity--;
}

public void RestockProduct(string code, int quantity)
{
    Product product = null;
    foreach (Product p in products)
    {
        if (p.Barcode == code)
        {
            product = p;
            break;
        }
    }
    product.Quantity += quantity;
}

```

```

public void PrintAllProducts()
{
    Console.WriteLine("Products:");
    foreach (Product product in products)
    {
        Console.WriteLine(product);
    }
}

public void PrintProductsToRestock()
{
    Console.WriteLine("Products to restock:");
    foreach (Product product in products)
    {
        if (product.Quantity < 10)
        {
            Console.WriteLine(product);
        }
    }
}
}
internal class Zad28
{
    static void Main(string[] args)
    {
        VendingMachine vendingMachine = new VendingMachine();
        using (System.IO.StreamReader reader = new
System.IO.StreamReader("data.txt"))
        {
            string line;
            while ((line = reader.ReadLine()) != "END")
            {
                try
                {
                    string[] command = line.Split();
                    switch (command[0])
                    {
                        case "AddDrink":
                            vendingMachine.AddProduct(new
Drink(command[1], command[2], int.Parse(command[3]),
double.Parse(command[4]), int.Parse(command[5])));
                            Console.WriteLine($"Added drink:
{command[2]} ({command[1]})");
                            break;
                        case "AddSnack":
                            vendingMachine.AddProduct(new
Snack(command[1], command[2], int.Parse(command[3]),
double.Parse(command[4]), int.Parse(command[5])));

```

```

                                Console.WriteLine($"Added      snack:
{command[2]} ({command[1]})");
                                break;
                                case "Buy":

vendingMachine.BuyProduct(command[1]);
                                Console.WriteLine($"Bought    product:
{command[1]}");
                                break;
                                case "Restock":

vendingMachine.RestockProduct(command[1], int.Parse(command[2]));
                                Console.WriteLine($"Restocked
product: {command[1]}, quantity: {int.Parse(command[2])}");
                                break;
                                case "PrintAll":
                                    vendingMachine.PrintAllProducts();
                                    break;
                                case "PrintLow":

vendingMachine.PrintProductsToRestock();
                                    break;
                                default:
                                    Console.WriteLine("Invalid command: "
+ command[0]);
                                    break;
                                }
                            }
                        catch (Exception e)
                        {
                            Console.WriteLine("Error: " + e.Message);
                        }
                    }
                Console.WriteLine("Goodbye!");
            }
        }
    }
}

```

3a Java

```

package DZI2026;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Task28 {

```

```

public static void main(String[] args) {
    VendingMachine vendingMachine = new VendingMachine();
    try (Scanner reader = new Scanner(new File("data.txt"))) {
        String line;
        while (reader.hasNextLine()) {
            line = reader.nextLine();
            if (line.equals("END")) {
                break;
            }
            try {
                String[] command = line.split(" ");
                switch (command[0]) {
                    case "AddDrink":
                        vendingMachine.addProduct(new
Drink(command[1],      command[2],      Integer.parseInt(command[3]),
Double.parseDouble(command[4]), Integer.parseInt(command[5])));
                        System.out.println("Added drink: " +
command[2] + " (" + command[1] + ")");
                        break;
                    case "AddSnack":
                        vendingMachine.addProduct(new
Snack(command[1],      command[2],      Integer.parseInt(command[3]),
Double.parseDouble(command[4]), Integer.parseInt(command[5])));
                        System.out.println("Added snack: " +
command[2] + " (" + command[1] + ")");
                        break;
                    case "Buy":
                        vendingMachine.buyProduct(command[1]);
                        System.out.println("Bought product: " +
command[1]);
                        break;
                    case "Restock":
                        vendingMachine.restockProduct(command[1],
Integer.parseInt(command[2]));
                        System.out.println("Restocked product: "
+ command[1] + ", quantity: " + command[2]);
                        break;
                    case "PrintAll":
                        vendingMachine.printAllProducts();
                        break;
                    case "PrintLow":
                        vendingMachine.printProductsToRestock();
                        break;
                    default:
                        System.out.println("Invalid command: " +
command[0]);
                }
            } catch (Exception e) {
                System.out.println("Error: " + e.getMessage());
            }
        }
    }
}

```

```

        }
    }
    System.out.println("Goodbye!");
}
}
}

```

```

abstract class Product {

    private String barcode;
    private String name;
    private int quantity;
    private double price;

    public String getBarcode() {
        return barcode;
    }

    public void setBarcode(String barcode) {
        if (barcode.length() != 10) {
            throw new IllegalArgumentException("Invalid barcode!");
        }
        this.barcode = barcode;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
        if (quantity < 0) {
            throw new IllegalArgumentException("Invalid quantity!");
        }
        this.quantity = quantity;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {

```

```

        this.price = price;
    }

    public Product(String barcode, String name, int quantity, double
price) {
        setBarcode(barcode);
        setName(name);
        setQuantity(quantity);
        setPrice(price);
    }

    @Override
    public String toString() {
        return String.format("%s %s - %d; %.2f eur", barcode, name,
quantity, price);
    }
}

class Drink extends Product {

    private int volume;

    public int getVolume() {
        return volume;
    }

    public void setVolume(int volume) {
        this.volume = volume;
    }

    public Drink(String barcode, String name, int quantity, double
price, int volume) {
        super(barcode, name, quantity, price);
        setVolume(volume);
    }

    @Override
    public String toString() {
        return super.toString() + String.format("; %d ml.", volume);
    }
}

class Snack extends Product {

    private int weight;

    public int getWeight() {
        return weight;
    }
}

```

```

    public void setWeight(int weight) {
        this.weight = weight;
    }

    public Snack(String barcode, String name, int quantity, double
price, int weight) {
        super(barcode, name, quantity, price);
        setWeight(weight);
    }

    @Override
    public String toString() {
        return super.toString() + String.format("; %d gr.", weight);
    }
}

class VendingMachine {

    private List<Product> products;

    public VendingMachine() {
        products = new ArrayList<>();
    }

    public void addProduct(Product product) {
        for (Product p : products) {
            if (p.getBarcode().equals(product.getBarcode())) {
                throw new IllegalArgumentException("Product with this
code already exists.");
            }
        }
        products.add(product);
    }

    public void buyProduct(String code) {
        Product product = null;
        for (Product p : products) {
            if (p.getBarcode().equals(code)) {
                product = p;
            }
        }
        if (product.getQuantity() == 0) {
            throw new IllegalArgumentException("Product is out of
stock.");
        }
        product.setQuantity(product.getQuantity() - 1);
    }
}

```

```

public void restockProduct(String code, int quantity) {
    Product product = null;
    for (Product p : products) {
        if (p.getBarcode().equals(code)) {
            product = p;
        }
    }
    product.setQuantity(product.getQuantity() + quantity);
}

public void printAllProducts() {
    System.out.println("Products:");
    for (Product product : products) {
        System.out.println(product);
    }
}

public void printProductsToRestock() {
    System.out.println("Products to restock:");
    for (Product product : products) {
        if (product.getQuantity() < 10) {
            System.out.println(product);
        }
    }
}
}

```