# C# OOP Regular Exam – 08 Dec 2024

A black paper with a red background

Description automatically generated

1. **Overview**

*Welcome to the BlackFriday Sales System! This application simulates a system for managing products and users during a Black Friday sale. It allows for registering different types of products and users, handling product discounts, and tracking purchases.*

*The system is built around key object-oriented programming concepts, using interfaces, inheritance, and encapsulation to ensure a well-structured and maintainable codebase. Administrators will oversee discounts and product management, while clients can purchase different items and services, creating a dynamic and interactive experience.*

## Setup

* Upload **only the BlackFriday** project in every task **except** **Unit Tests.**
* **Do not modify the interfaces or their packages.**
* Use **strong cohesion** and **loose coupling.**
* **Use inheritance and the provided interfaces wherever possible**:
  + This includes **constructors**, **method parameters,** and **return types.**
* **Do not** violate your **interface** **implementations** by adding **more public methods** in the concrete class than the interface has defined.
* Make sure you have **no public fields** anywhere.
* **Exception messages** and **output messages** can be found in the **"Utilities"** folder.
* To solve this problem use **Visual Studio 2022** and **net 6.0**
* **Do not use** "\r\n" **for a new line.**

## Task 1: Structure (50 points)

**For this task’s evaluation logic in the methods isn’t included.**

You are given some **interfaces**, and you have to **implement** their functionality in the **correct classes**.

There are **3** types of entities: **Product**, **User**, and **Application**. You should also implement two generic repositories: **ProductRepository** and **UserRepository**.

### Product

The Product is a **base class** of any type of product**,** and it **should not be able to be instantiated directly**.

#### Data

* **ProductName** - **string**
  + Must NOT be **null or whitespace.** If **invalid** throw a new **ArgumentException** with the message:

"Product name is required."

* **BasePrice –** **double**
  + Must be a **positive number**. If **invalid**, throw a new **ArgumentException** with the message:   
    "Price cannot be zero or negative."
* **BlackFridayPrice – double**
  + A virtual property, **overridden** in child classes to calculate the discounted price.
* **IsSold – bool**
  + Indicates whether the product has been sold. Set to false as default.

#### Override ToString() method:

Overrides the existing method ToString()and modify it, so the returned **string must be on a single line**, in the following format:

**"**Product: **{ProductName}**, Price: **{BasePrice:F2}**, You Save: **{(BasePrice-BlackFridayPrice):F2}"**

#### Constructor

A Product should take the following values upon initialization:

string productName, double basePrice

* The constructor validates and sets the ProductName and BasePrice properties.
* Initializes the IsSold property to false

#### Behavior

##### void UpdatePrice(double newPriceValue)

Updates the product's base price with a new value.

##### void ToggleStatus()

* Marks the IsSold property to true if it is marked as false OR to false if it is marked as true.

#### Child Classes

There are two specific types of Product:

**Item**

Overrides the BlackFridayPrice property to return a value with a **30% discount** on the BasePrice

The Constructor of the **Item** should take the following parameters upon initialization:

string productName, double basePrice

**Service**

Overrides the BlackFridayPrice property to return a value with a **20% discount** on the BasePrice

The Constructor of the **Service** should take the following parameters upon initialization:

string productName, double basePrice

### User

The User is a **base class** of any **type of user,** and it **should not be able to be instantiated directly**.

#### Data

* **UserName** - **string**
  + If the **name** is **null or whitespace,** throw a new **ArgumentException** with the message:

"Username is required."

* HasDataAcess – bool
  + This property indicates whether the user has access to data management features
  + Its behavior is **determined by the specific child class: for Admin always returns true / for User always returns false**
  + The value is **not directly set in the base class** but is **implemented differently in each child class**.
* **Email – string**
  + If HasDataAccess is true, **no exception is thrown for a missing or invalid email**. In this case the property returns: "hidden"
  + If HasDataAccess is false, Email is **validated properly. If null or whitespace is passed,** throw a new **ArgumentException** with the message: "Email is required."

#### Override ToString() method:

Overrides the existing method ToString()and modifies it, so the returned string **must be on a single line**, in the following format:

**"{UserName}** - Status: **{**"Client/Admin"**}**, Contact Info: **{email}"**

Depends on the type name

#### Constructor

A **User** should take the following values upon initialization:

string userName, string email, bool hasDataAccess

#### Child Classes

There are two concrete types of **User**:

##### Client

HasDataAccess property returns false

Purchases – IReadOnlyDictionary<string, bool>

* **Additional property** holding:
  + **Keys -**  the names of all purchased products
  + **Values –** if the **product is bought with a discount**, set the value to true, otherwise - false

The Constructor of the **Client** should take the following parameters upon initialization:

stringuserName, stringemail

* Also initializes the Purchases collection

#### Behavior

##### void PurchaseProduct(string productName, bool blackFridayFlag)

Adds the productName as a **Key** to the Purchases collection, with **Value** - blackFridayFlag

##### Admin

HasDataAccess property returns true

The Constructor of the **Admin** should take the following parameters upon initialization:

stringuserName, stringemail

## ProductRepository

The **ProductRepository** is an **IRepository<IProduct>. Collection** of all existing **products in the application**.

### Data

* **Models – a collection of products (unmodifiable)**

### Behavior

**void AddNew(IProduct product)**

* **Adds** a **product** to the repository.

**IProduct GetByName(string name)**

* Returns a product with a **productName value, equal to the given parameter**. If **no such product is found** in the repository **returns null**.

**bool Exists(string name)**

* Returns **true** if a product with a **productName matching the given parameter exists** in the repository,   
  **otherwise returns false**.

## UserRepository

The **UserRepository** is an **IRepository<IUser>. Collection** of all existing **users in the application**.

### Data

* **Models – a collection of users (unmodifiable)**

### Behavior

**void AddNew(IUser user)**

* **Adds** a **user** to the repository.

**IUser GetByName(string name)**

* Returns a user with a **userName value, equal to the given parameter**. No such user - **returns null**.

**bool Exists(string name)**

* Returns **true** if a user with a **userName matching the given parameter exists** in the repository,   
  **otherwise returns false**.

## Application

### Data

* **Products – IRepository<IProduct>**
* **Users – IRepository<IUser>**

### Constructor

An **Application** should take **no values** upon initialization and should **initialize new instances** of the **Products** and **Users** collections.

## Task 2: Business Logic (150 points)

## The Controller Class

The business logic of the program should be concentrated around several **commands**. You are given interfaces, which you have to implement in the correct classes.

**NOTE: Do not use** "\r\n" **for a new line.**

The first interface is **IController**. You must create a **Controller** class, which implements the interface and implements all of its methods. The constructor of **Controller** does not take any arguments. The given methods should have the logic described for each in the Commands section. When you create the **Controller** class, go into the **Engine** class constructor and uncomment the "this.controller = new Controller();" line.

**Data**

You need to keep track of some collections, this is why you need a private field in your controller class:

**Example:**

* **application – IApplication** (you can name it as you like, **Judge will not test this property**)

**Commands**

There are several commands, which control the business logic. They are stated below.

#### RegisterUser Command

##### Parameters

* **userName – string**
* **email – string**
* **hasDataAccess - bool**

##### Functionality

The method should **create and add** a new **IUser** to the appropriate **application's** collection.

* If the given userName is already registered in the application, skip the registration process and return the following message: "{userName**}** is already registered."
* If any user has made a registration with the same email, skip the registration process and return the following message: "{**email}** is already used by another user."
* If the hasDataAccess parameter is true, you should register an Admin user. Only 2 admin roles are allowed in the application.
  + If there are two admins already registered a third one will be not allowed. Skip the registration process and return:  
    "The number of application administrators is limited."
  + If the above case is not reached, create the appropriate IUser and add it to the appropriate application's repository. The following message is returned:  
    "Admin {**userName}** is successfully registered with data access."
* If the hasDataAccess parameter is false, a new Client user is added to the appropriate application's repository. The following message is returned:   
  "Client {**userName}** is successfully registered."

#### AddProduct Command

##### Parameters

* **productType - string**
* **productName – string**
* **userName - string**
* **basePrice - double**

##### Functionality

The method should **create and add** a new **IProduct** to the appropriate application's collection.

* If the given productType is NOT presented as a valid **Product's** child class (Item, Service), return the following message:   
  "{productType**}** is not a valid type for the application."
* If any product from the with the same **productName** is already added to the application, do not duplicate records, return the following message:   
  "{**productName}** already exists in the application."
* If the **user with** the givenusername is not registered **OR** is just a Client, he has no data access and **won't be allowed** to add a new IProduct to the application. The following message is returned:  
  "{userName} has no data access.**"**
* If none of the above cases is reached, the IProduct is successfully created. Add the product to the appropriate application's collection and return: "{productType}: {**productName}** is added in the application. Price: {**basePrice:F2}**"

#### UpdateProductPrice Command

##### Parameters

* **productName – string**
* **userName - string**
* **newPriceValue - double**

##### Functionality

The method should **set a new** basePricevalue to the **IProduct** that has ProductName matching the given productName parameter.

* If a product with the same name is not found in the application, return the following message:   
  "{**productName}** does not exist in the application."
* If the **user with** the givenusername is not registered **OR** is just a Client, he has no data access and **won't be allowed** to update a product's price. The following message is returned:  
  "{userName} has no data access.**"**
* If none of the above cases is reached, you should extract the product with the given productName from the application and update its price. The following message is returned:   
  "{productName} -> Price is updated: {**oldPriceValue:F2}** -> {**newPriceValue:F2}**"

#### PurchaseProduct Command

##### Parameters

* **userName - string**
* **productName – string**
* **blackFridayFlag - bool**

##### Functionality

The method should **allow a user to purchase a specific product**.

* If the **user with** the givenusername is not registered **OR** is an Admin, he has no access to buy a product and **won't be allowed** to. The following message is returned:   
  "{userName} has no authorization for this functionality.**"**
* If a product with the same name is not found in the application, return the following message:   
  "{**productName}** does not exist in the application."
* If a product with the same name exists in the application, but is sold currently, return the following message: "{**productName}** is out of stock."
* If none of the above cases is reached, you should extract the product with the given productName from the application and
  + add its productName to the Client's Purchases collection(use PurchaseProduct method of the Client and pass blackFridayFlag when adding to the Purchases collection)
  + set its IsSold property to true
* If the blackFridayFlag is true, the product is purchased on BlackFridayPrice. Otherwise, it is purchased on BasePrice. Return the following message:   
  "{userName} purchased {**productName}**. Price: {**price:F2}**"

#### RefreshSalesList Command

##### Parameters

* **userName - string**

##### Functionality

The method should **extract all products** that **are sold** and **toggle** theirIsSoldproperty.

* If the **user with** the givenusername is not registered **OR** is just a Client, he has no data access and **won't be allowed** to update a product's price. The following message is returned:  
  "{userName} has no data access.**"**
* If the above case is not reached, you should extract all products with IsSold property == true and change it to false. The following message is returned:   
  "{**updatedProductsCount}** products are listed again."

#### ApplicationReport Command

##### Functionality

This method generates a **detailed report for all users of the application**, providing insights into their purchases. The report is structured as follows:

* Admins appear first, **ordered by** their **names**, **alphabetically**
* Clients are **ordered by** their **names**, **alphabetically**
* **For each client**, the **products they have purchased with BlackFriday discount**. The **count of purchases made** with the **blackFridayFlag** set to **true**.
* If a **customer has not purchased any products with a BlackFriday discount**, **skip** **the whole**"--Black Friday Purchases:" **section**

"Application administration:

There will always be at least one admin added to the application

**{admin1}**

**{admin2}**

Clients:

**{client1}**

-Black Friday Purchases: **{count}**

If the Client has made **any Black Friday purchases**, print details about the product here, using the product's name from the Purchases collection, and extracting information from the ProductRepository

-**{product1}**

-**{product2}**

**…**

-**{productn}**

**{client2}**

**…**

**{clientn}**

**NOTE: Do not use** "\r\n" **for a new line.**

#### Exit Command

##### Functionality

Ends the program.

### Input / Output

You are provided with one interface, which will help you with the correct execution process of your program. The interface is Engine, and the class implementing this interface should read the input, and when the program finishes, this class should print the output.

#### Input

Below, you can see the **format** in which **each command** will be given in the input:

* **RegisterUser {userName} {email} {hasDataAccess}**
* **AddProduct {productType} {productName} {username} {basePrice}**
* **UpdateProductPrice {productName} {userName} {newPriceValue}**
* **PurchaseProduct {userName} {productName} {blackFridayFlag}**
* **RefreshSalesList {userName}**
* **ApplicationReport**
* **Exit**

#### Output

Print the output from each command when issued. Print the exception message if an exception is thrown during any of the commands' execution.

#### Examples

|  |
| --- |
| **Input** |
| **RegisterUser CommonUser user@applicationmail.bg false**  **RegisterUser CommonUser admin@applicationmail.bg false**  **RegisterUser AdminUser1 admin\_one@applicationmail.bg true**  **RegisterUser OtherCommonUser user@applicationmail.bg false**  **RegisterUser AdminUser2 admin\_two@applicationmail.bg true**  **RegisterUser AdminUser3 admin\_three@applicationmail.bg true**  **RegisterUser john\_doe john.doe@gmail.com false**  **RegisterUser MikeRoss mike.ross@hotmail.com false**  **RegisterUser janeSVG jennifer\_savage@mail.de false**  **AddProduct Item Laptop AdminUser1 1200.00**  **AddProduct Service Warranty CommonUser 200.00**  **AddProduct Item Laptop AdminUser1 1500.00**  **AddProduct Furniture Desk AdminUser2 350.00**  **AddProduct Item Phone UnknownUser 900.00**  **AddProduct Service SoftwareInstallation AdminUser2 100.00**  **AddProduct Item Monitor AdminUser1 300.00**  **AddProduct Item GamingMouse AdminUser2 80.00**  **AddProduct Item ExternalHardDrive AdminUser1 120.00**  **AddProduct Service AntivirusSubscrition AdminUser2 50.00**  **AddProduct Service CloudStorage AdminUser1 10.00**  **AddProduct Service DeviceSetup AdminUser2 40.00**  **UpdateProductPrice Laptop AdminUser1 1300.00**  **UpdateProductPrice Tablet AdminUser1 500.00**  **UpdateProductPrice Laptop john\_doe 1250.00**  **UpdateProductPrice Monitor UnknownUser 350.00**  **UpdateProductPrice CloudStorage AdminUser2 15.00**  **PurchaseProduct john\_doe Laptop true**  **PurchaseProduct MikeRoss Monitor false**  **PurchaseProduct AdminUser1 GamingMouse false**  **PurchaseProduct janeSVG SmartWatch true**  **PurchaseProduct john\_doe Laptop false**  **PurchaseProduct UnknownUser DeviceSetup true**  **PurchaseProduct MikeRoss CloudStorage true**  **PurchaseProduct john\_doe GamingMouse true**  **PurchaseProduct MikeRoss CloudStorage true**  **PurchaseProduct john\_doe SoftwareInstallation true**  **PurchaseProduct MikeRoss AntivirusSubscrition true**  **PurchaseProduct john\_doe ExternalHardDrive true**  **RefreshSalesList janeSVG**  **RefreshSalesList AdminUser1**  **PurchaseProduct janeSVG CloudStorage true**  **ApplicationReport**  **Exit** |
| **Output** |
| **Client CommonUser is successfully registered.**  **CommonUser is already registered.**  **Admin AdminUser1 is successfully registered with data access.**  **user@applicationmail.bg is already used by another user.**  **Admin AdminUser2 is successfully registered with data access.**  **The number of application administrators is limited.**  **Client john\_doe is successfully registered.**  **Client MikeRoss is successfully registered.**  **Client janeSVG is successfully registered.**  **Item: Laptop is added in the application. Price: 1200.00**  **CommonUser has no data access.**  **Laptop already exists in the application.**  **Furniture is not a valid type for the application.**  **UnknownUser has no data access.**  **Service: SoftwareInstallation is added in the application. Price: 100.00**  **Item: Monitor is added in the application. Price: 300.00**  **Item: GamingMouse is added in the application. Price: 80.00**  **Item: ExternalHardDrive is added in the application. Price: 120.00**  **Service: AntivirusSubscrition is added in the application. Price: 50.00**  **Service: CloudStorage is added in the application. Price: 10.00**  **Service: DeviceSetup is added in the application. Price: 40.00**  **Laptop -> Price is updated: 1200.00 -> 1300.00**  **Tablet does not exist in the application.**  **john\_doe has no data access.**  **UnknownUser has no data access.**  **CloudStorage -> Price is updated: 10.00 -> 15.00**  **john\_doe purchased Laptop. Price: 910.00**  **MikeRoss purchased Monitor. Price: 300.00**  **AdminUser1 has no authorization for this functionality.**  **SmartWatch does not exist in the application.**  **Laptop is out of stock.**  **UnknownUser has no authorization for this functionality.**  **MikeRoss purchased CloudStorage. Price: 12.00**  **john\_doe purchased GamingMouse. Price: 56.00**  **CloudStorage is out of stock.**  **john\_doe purchased SoftwareInstallation. Price: 80.00**  **MikeRoss purchased AntivirusSubscrition. Price: 40.00**  **john\_doe purchased ExternalHardDrive. Price: 84.00**  **janeSVG has no data access.**  **7 products are listed again.**  **janeSVG purchased CloudStorage. Price: 12.00**  **Application administration:**  **AdminUser1 - Status: Admin, Contact Info: hidden**  **AdminUser2 - Status: Admin, Contact Info: hidden**  **Clients:**  **CommonUser - Status: Client, Contact Info: user@applicationmail.bg**  **janeSVG - Status: Client, Contact Info: jennifer\_savage@mail.de**  **-Black Friday Purchases: 1**  **--CloudStorage**  **john\_doe - Status: Client, Contact Info: john.doe@gmail.com**  **-Black Friday Purchases: 4**  **--Laptop**  **--GamingMouse**  **--SoftwareInstallation**  **--ExternalHardDrive**  **MikeRoss - Status: Client, Contact Info: mike.ross@hotmail.com**  **-Black Friday Purchases: 2**  **--CloudStorage**  **--AntivirusSubscrition** |

## Task 3: Unit Tests (100 points)

You will receive a **NetTraderSystem** skeleton with **Product** and **TradingPlatform** classes inside. **TradingPlatform** class will have some methods, fields, and constructors, that are working properly You are **NOT ALLOWED** to change any class. **Cover the whole class** (only **TradingPlatform**) with unit tests to make sure that the class is working as intended. You are provided with a **unit test project** in the **project skeleton**.

* **Do NOT CHANGE OR REMOVE ANY namespaces or usings.**
* **Do not use** "\r\n" **for a new line.**
* Do **NOT** use **Mocking** in your unit tests!
* In Judge, you upload **.zip** of the **unit test project** from the **skeleton.**

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

A blue box with black text

Description automatically generated A screenshot of a computer

Description automatically generated