## OOP Sample Exam 2 – 1. Cosmetics Shop

We need to create an online cosmetics shop.

In the shop there are currently two types of **products**: **shampoos** and **toothpastes**. Each **product** has **name**, **brand**, **price** and **gender** (men, women, unisex). Each **shampoo** has **quantity** (in milliliters) and **usage** (every day or medical). All shampoos’ prices are per milliliter. Toothpastes have **ingredients**. There are **categories** of products. Each **category** has **name** and products can be **added** or **removed**. The same product can be added to a category more than once. There is also a **shopping cart**. Products can be **added** or **removed** from it. The same product can be added to the shopping cart more than once. The shopping cart can calculate the **total price** of all products in it.

### Design the Class Hierarchy

Your **task** is to **design an object-oriented class hierarchy** to model the cosmetics shop, **using the best practices of object-oriented design (OOD) and object-oriented programming (OOP)**. Avoid duplicated code though abstraction, inheritance and polymorphism and encapsulate correctly all fields.

You are given a few C# **interfaces** that **you are obligated to implement** and use as a basis of your code:

|  |
| --- |
| namespace Cosmetics.Contracts  {  public interface ICategory  {  string Name { get; }  void AddCosmetics(IProduct cosmetics);  void RemoveCosmetics(IProduct cosmetics);  string Print();  }  public interface ICosmeticsFactory  {  ICategory CreateCategory(string name);  IShampoo CreateShampoo(string name, string brand, decimal price, GenderType gender,  uint milliliters, UsageType usage);  IToothpaste CreateToothpaste(string name, string brand, decimal price,  GenderType gender, IList<string> ingredients);  IShoppingCart ShoppingCart();  }  public interface IProduct  {  string Name { get; }  string Brand { get; }  decimal Price { get; }  GenderType Gender { get; }  string Print();  }  public interface IShampoo : IProduct  {  uint Milliliters { get; }  UsageType Usage { get; }  }  public interface IToothpaste : IProduct  {  string Ingredients { get; }  }  public interface IShoppingCart  {  void AddProduct(IProduct product);  void RemoveProduct(IProduct product);  bool ContainsProduct(IProduct product);  decimal TotalPrice();  }  } |

Categories should implement **ICategory**. Adding the same product to one category more than once is allowed. Minimum category name length is 2 symbols and maximum is 15 symbols. The error message should be **"Category name must be between {min} and {max} symbols long!"** Products in a category should be **sorted** by brand in ascending order and then by price in descending order. When removing a product from a category, if the product is not found, the error message should be **"Product {product name} does not exist in category {category name}!"** Category’s print method should return text in the following format:

|  |
| --- |
| ***{category name} category – {number of products} products/product in total***  ***- {product brand} – {product name}:***  ***\* Price: ${product price}***  ***\* For gender: Men/Women/Unisex***  ***\* Ingredients: {product ingredients, separated by “, “}* (when applicable)**  ***- {product brand} – {product name}:***  ***\* Price: ${product price}***  ***\* For gender: {product gender}***  ***\* Quantity: {product quantity} ml* (when applicable)**  ***\* Usage: EveryDay/Medical* (when applicable)** |

All products should implement the **IProduct** interface. Minimum product name length is 3 symbols and maximum is 10 symbols. The error message should be **"Product name must be between {min} and {max} symbols long!"** Minimum brand name length is 2 symbols and maximum is 10 symbols. The error message should be **"****Product brand must be between {min} and {max} symbols long!"** Gender type can be "Men", "Women" or "Unisex".

All shampoos should implement the **IShampoo** interface. Shampoo’s price is given per milliliter. Usage type can be "EveryDay" or "Medical".

All toothpastes should implement the **IToothpaste** interface. Ingredients should be represented as text, joined in their order of addition, separated by ", " (comma and space). Each ingredient’s name length should be between 4 and 12 symbols, inclusive. The error message should be **"Each ingredient must be between {min} and {max} symbols long!"**

Shopping cart should implement the **IShoppingCart** interface. Adding the same product more than once is allowed. Do not check if the product exists when removing it from the shopping cart.

Check out the example to get a better understanding of the printing format.

All number type fields should be printed "**as is**", without any formatting or rounding.

All properties in the above interfaces are mandatory (cannot be null or empty).

If a null value is passed to some mandatory property, your program should throw a proper exception.

### Additional Notes

To simplify your work you are given an already built execution engine that executes a sequence of commands read from the console using the classes and interfaces in your project (see the Cosmetics-Skeleton folder). Please put your classes in the namespace **Cosmetics.Products**. Implement the **CosmeticsFactory** class in the namespace **Cosmetics.Engine**.

You are only **allowed to write classes**. You are **not allowed to modify the existing interfaces and classes except the CosmeticsFactory class**.

Current implemented commands the engine supports are:

* **CreateCategory (name)** – adds a category with a given name. Duplicate names are not allowed
* **AddToCategory (categoryName) (productName)** – adds a product to a category, if both are already created in the program
* **RemoveFromCategory (categoryName) (productName)** – removes a product from a category, if both are already created in the program
* **ShowCategory (categoryName)** – prints the category and all products in it
* **CreateShampoo (name) (brand) (price) (gender) (milliliters) (usage)** – parses the input and creates a shampoo. Duplicate names are not allowed
* **CreateToothpaste (name) (brand) (price) (gender) (ingredients) –** parses the input and creates a toothpaste. Ingredients are comma separated. Duplicate names are not allowed
* **AddToShoppingCart (productName) –** adds a product to the shopping cart, if the product is already created
* **RemoveFromShoppingCart (productName) –** removes a product from the shopping cart, if the product is created and is already in the shopping cart
* **TotalPrice –** returns the total price of all products in the shopping cart

All commands return appropriate success messages. In case of invalid operation or error, the engine returns appropriate error messages.