Exercício 1 - Delegates

Inicialmente, crie uma lista contendo os seguintes elementos do tipo `double`: 3, 7, 2, 4, 6. Na sequência, chame o método `ConvertAll()` da lista para retornar uma nova lista de elementos `double`, onde cada um dos elementos originais é dividido por 2. Isto significa que a nova lista conterá os elementos 1.5, 3.5, 1, 2, 3.

O método `ConvertAll()` recebe como parâmetro um delegate do tipo `Converter`. Este delegate é responsável por transformar cada um dos elementos da lista, a fim de serem adicionados na nova lista. Use uma expressão lambda, que recebe como parâmetro o elemento a ser transformado.

Depois que esta nova lista for gerada, imprima os elementos na tela. A impressão deve ser feita através do método `ForEach()` da lista, que recebe um delegate do tipo `Action`. Aqui também você deve usar uma expressão lambda, que recebe como parâmetro o elemento que será exibido na tela.

Exercício 2 – Linq

Crie uma classe `ItemMercado`, cujos objetos correspondem a itens que você precisa comprar no supermercado. A classe deve possuir as properties `Nome` (`string`), `Tipo` (`Tipo`) e `Preco` (`double`). `Tipo` deve ser um enum, que pode ter os valores: `Comida`, `Bebida`, `Higiene` e `Limpeza`.

Na sequência, crie uma lista de objetos do tipo `ItemMercado`, de acordo com a tabela abaixo:

Nome	Tipo	Preço
Arroz	Comida	R$ 3,90
Azeite	Comida	R$ 2,50
Macarrão	Comida	R$ 3,90
Cerveja	Bebida	R$ 22,90
Refrigerante	Bebida	R$ 5,50
Shampoo	Higiene	R$ 7,00
Sabonete	Higiene	R$ 2,40
Cotonete	Higiene	R$ 5,70
Sabão em pó	Limpeza	R$ 8,20
Detergente	Limpeza	R$ 2,60
Amaciante	Limpeza	R$ 6,40

Com base nos dados desta lista, crie uma série de expressões LINQ para retornar algumas informações:

1) Retorne uma lista de itens do tipo `Higiene` ordenados por ordem decrescente de preço.
2) Retorne uma lista de itens cujo preço seja maior ou igual a R$ 5,00. A ordenação deve ser feita por ordem crescente de preço.
3) Retorne uma lista de itens cujo tipo seja `Comida` ou `Bebida`. A ordenação deve ser feita por nome em ordem alfabética.
4) Retorne cada um dos tipos associado com a quantidade de itens de cada tipo.
5) Retorne cada um dos tipos associado com o preço máximo, preço mínimo e média de preço de cada tipo.

Exercício 3 - Threads

Vamos praticar a criação e execução de threads em C#. Vocês precisarão criar um programa que demonstre o uso de threads para realizar um trabalho simultâneo. O programa deve conter uma classe Worker com um método Work() que simule algum tipo de trabalho. Criem então duas instâncias dessa classe e as executem simultaneamente em duas threads diferentes. As threads devem imprimir mensagens indicando o progresso do trabalho que estão realizando. Por fim, o programa principal deve aguardar até que ambas as threads tenham terminado de executar antes de encerrar.

Exercício 4 – Async/Await

Utilizando async/await em C#

Vamos praticar o uso de métodos assíncronos em C# utilizando as palavras-chave async e await. Seu objetivo é criar um programa que demonstre a execução de duas tarefas simultâneas de forma assíncrona. Crie um programa em C# que contenha um método assíncrono chamado DoWorkAsync. Este método deve simular algum tipo de trabalho realizando um loop que imprime mensagens indicando o progresso do trabalho em andamento. Utilize Task.Delay para simular o tempo de espera entre cada iteração do loop.

No método Main, inicie duas tarefas simultâneas chamando o método DoWorkAsync. Uma tarefa deve representar a execução da "Tarefa 1" e a outra a execução da "Tarefa 2".

Aguarde a conclusão de ambas as tarefas de forma assíncrona utilizando Task.WhenAll.

Após a conclusão de ambas as tarefas, imprima uma mensagem indicando que ambas as tarefas foram concluídas. Teste o seu programa para garantir que as tarefas estão sendo executadas de forma assíncrona e que a espera está ocorrendo corretamente. Experimente com diferentes tempos de espera e números de iterações para entender melhor o comportamento do programa assíncrono.