Delegates , Lambda, LINQ

1- Considere as seguintes assertivas sobre delegates na linguagem C#. Indique se cada uma é verdadeira ou falsa:

a-) Um delegate é um tipo de dado que pode ser usado para encapsular um método e chamá-lo posteriormente.

b-) Para declarar um delegate em C#, é necessário especificar o tipo de retorno do método que ele encapsula, bem como os tipos de parâmetros.

c-) É possível usar o operador += para adicionar vários métodos a um delegate, que serão chamados em sequência quando o delegate for invocado.

d-) O delegate pode ser passado como parâmetro para outro método, permitindo que o método receptor invoque o método encapsulado no delegate.

e-) O uso de delegates pode melhorar a modularidade e a flexibilidade do código, permitindo que um método seja substituído por outro sem que isso afete o restante do código.

f-) Delegates só podem ser usados para encapsular métodos estáticos, não métodos de instância.

g-) O delegate é um recurso exclusivo da linguagem C#, e não pode ser encontrado em outras linguagens de programação.

h-) Delegates são frequentemente usados em eventos, para permitir que um objeto notifique outros objetos quando ocorre alguma ação.

Respostas:

a-) Verdadeira
b-) Verdadeira
c-) Verdadeira
d-) Verdadeira
e-) Verdadeira
f-) Falsa
g-) Falsa
h-) Verdadeira

Justificativa:

a-) Um delegate é exatamente isso: um tipo de dado que pode ser usado para referenciar e invocar um método posteriormente.
b-) Sim, é necessário especificar tanto o tipo de retorno quanto os tipos de parâmetros do método encapsulado pelo delegate.
c-) Sim, ao usar o operador +=, é possível adicionar vários métodos a um delegate, que serão chamados em sequência quando o delegate for invocado.
d-) Sim, é possível passar um delegate como parâmetro para outro método e invocá-lo dentro desse método.
e-) Sim, o uso de delegates permite que um método seja substituído por outro sem que isso afete o restante do código, tornando o código mais modular e flexível.
f-) Não, delegates podem encapsular tanto métodos estáticos quanto métodos de instância.
g-) Não, delegates também podem ser encontrados em outras linguagens de programação, como C++ e Java.
h-) Sim, delegates são frequentemente usados em eventos para permitir que um objeto notifique outros objetos quando ocorre alguma ação.

Delegates , Lambda, LINQ

2- Implemente um programa que use delegates para somar dois números inteiros. O programa deve seguir as seguintes especificações:

a-Defina um delegate chamado OperacaoMatematica que recebe dois inteiros como parâmetros e retorna um inteiro.
b-Defina um método que implementam a operação de soma: que recebe os dois inteiros como parâmetros
c-Crie um objeto delegate do tipo OperacaoMatematica e atribua a ele ao método definido.
d-Chame o delegate passando os parâmetros necessários e exiba o resultado da operação na tela.

Dica: Utilize o método Invoke do delegate para invocar a operação matemática.

Resposta:

```
OperacaoMatematica opMat = new OperacaoMatematica(Soma);

// Executa o delegate com os valores 5 e 7 e exibe o resultado
int resultado = opMat.Invoke(5, 7);
Console.WriteLine("Resultado da operação de soma: " + resultado);

Console.ReadKey();

static int Soma(int x, int y)
{
    return x + y;
}

delegate int OperacaoMatematica(int x, int y);
```

Neste exemplo, o delegate OperacaoMatematica é definido para receber dois inteiros como parâmetros e retornar um inteiro.
um objeto delegate do tipo OperacaoMatematica é criado e atribuído ao método Soma
O delegate é então chamado com os parâmetros 5 e 7, e o resultado é exibido na tela.

3- Implemente um programa em C# que use multicast delegates para exibir uma mensagem de boas-vindas ao usuário. O programa deve seguir as seguintes especificações:

a-) Defina um multicast delegate chamado MensagemBoasVindas que não recebe parâmetros e não retorna nenhum valor.
b-) Defina duas funções que exibem uma mensagem de boas-vindas: uma em português e outra em inglês.
c-) Crie um objeto delegate do tipo MensagemBoasVindas e adicione as duas funções criadas a ele usando o operador +=.
d-) Chame o delegate para executar as duas funções e exibir as mensagens de boas-vindas.

Delegates , Lambda, LINQ

Resposta:

```
MensagemBoasVindas mensagem = new MensagemBoasVindas(BoasVindasEmPortugues);

mensagem += BoasVindasEmIngles;
mensagem.Invoke();

Console.ReadKey();

static void BoasVindasEmPortugues()
{
    Console.WriteLine("Bem-vindo!");
}
static void BoasVindasEmIngles()
{
    Console.WriteLine("Welcome!");
}
delegate void MensagemBoasVindas();
```

Neste exemplo, o multicast delegate MensagemBoasVindas é definido para não receber parâmetros e não retornar nenhum valor.
Em seguida, dois métodos exibem mensagens de boas-vindas são definidas: BoasVindasEmPortugues e BoasVindasEmIngles.
No método Main, um objeto delegate do tipo MensagemBoasVindas é criado e as duas funções são adicionadas a ele usando o operador +=.
O delegate é então chamado usando o método Invoke, e as duas funções são executadas, exibindo as mensagens de boas-vindas em português e em inglês.

4- Dado o código abaixo, assinale com verdadeiro ou falso cada afirmação sobre os delegates Action, Predicate e Func da linguagem C#:

```
int[] numeros = { 1, 2, 3, 4, 5 };

// Delegate Action
Action<int> exibirNumero = num => Console.WriteLine(num);
Array.ForEach(numeros, exibirNumero);

// Delegate Predicate
Predicate<int> ehPar = num => num % 2 == 0;
bool todosPares = Array.TrueForAll(numeros, ehPar);

// Delegate Func
Func<int, int, int> somar = (a, b) => a + b;
int resultado = somar(10, 20);

Console.WriteLine(todosPares);
Console.WriteLine(resultado);

Console.ReadKey();
```

a- O delegate Action é usado para encapsular um método que recebe um parâmetro e não retorna nenhum valor. (Verdadeiro ou Falso)
b- O método Array.ForEach é usado para percorrer todos os elementos de um array e executar um delegate para cada um deles. (Verdadeiro ou Falso)
c- O delegate Predicate é usado para encapsular um método que recebe um parâmetro e retorna um valor booleano. (Verdadeiro ou Falso)
d- O método Array.TrueForAll retorna true se todos os elementos de um array satisfazem uma condição especificada por um delegate. (Verdadeiro ou Falso)

e- O delegate Func é usado para encapsular um método que recebe um ou mais parâmetros e retorna um valor. (Verdadeiro ou Falso)

f- O delegate Func pode ter até 16 parâmetros de entrada e um tipo de retorno genérico. (Verdadeiro ou Falso)

g- No exemplo acima, o delegate Func é usado para definir um método que soma dois números inteiros. (Verdadeiro ou Falso)

h- No exemplo acima, o método somar recebe dois parâmetros e retorna um valor. (Verdadeiro ou Falso)

Respostas:

a- Verdadeiro
b- Verdadeiro
c- Verdadeiro
d- Verdadeiro
e- Verdadeiro
f- Verdadeiro
g- Verdadeiro
h- Verdadeiro

a- Verdadeiro. O delegate Action é usado para encapsular um método que não retorna valor e recebe um parâmetro. O método exibirNumero é um delegate Action que recebe um inteiro como parâmetro e imprime o valor na tela.

b- Verdadeiro. O método Array.ForEach percorre todos os elementos de um array e executa um delegate para cada um deles. O segundo parâmetro do método é um delegate Action que será executado para cada elemento do array.

c- Verdadeiro. O delegate Predicate é usado para encapsular um método que recebe um parâmetro e retorna um valor booleano. O método ehPar é um delegate Predicate que retorna true se o número passado como parâmetro é par e false caso contrário.

d- Verdadeiro. O método Array.TrueForAll retorna true se todos os elementos de um array satisfazem uma condição especificada por um delegate. O segundo parâmetro do método é um delegate Predicate que define a condição a ser verificada para cada elemento do array.

e- Verdadeiro. O delegate Func é usado para encapsular um método que recebe um ou mais parâmetros e retorna um valor. O método somar é um delegate Func que recebe dois inteiros como parâmetros e retorna a soma desses valores.

f- Verdadeiro. O delegate Func pode ter até 16 parâmetros de entrada e um tipo de retorno genérico.

g- Verdadeiro. No exemplo acima, o delegate Func é usado para definir um método que soma dois números inteiros. O método somar é um delegate Func que recebe dois inteiros como parâmetros e retorna a soma desses valores.

h- Verdadeiro. No exemplo acima, o método somar recebe dois parâmetros do tipo inteiro e retorna um valor do tipo inteiro que é a soma desses dois valores.

5- Imagine que você precisa criar um método que imprima na tela os números pares de uma lista de inteiros de 1 a 20.

Para isso, você deve utilizar um delegate Action<int> que receba um número inteiro como parâmetro e imprima na tela apenas os números pares.

Resposta:

```
List<int> numeros = new List<int> { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

Action<int> imprimirPares = (num) => {
    if (num % 2 == 0)
    {
        Console.WriteLine(num);
    }
};
numeros.ForEach(imprimirPares);

Console.ReadKey();
```

Nesse exemplo, o método Main cria uma lista de inteiros e define um delegate Action<int> chamado imprimirPares que verifica se um número é par e imprime na tela caso seja.
O método ForEach da lista de inteiros recebe o delegate imprimirPares como parâmetro e executa o método para cada elemento da lista. Assim, apenas os números pares são impressos na tela

6- Crie um programa onde um método que receba dois números inteiros e retorne o resultado da operação de adição ou subtração desses números, a ser definida em tempo de execução por um delegate Func<int, int, int>.

Resposta:

```
int numero1 = 5;
int numero2 = 3;

Func<int, int, int> operacao = (a, b) => a + b;

Console.WriteLine($"O resultado da operação é {Operar(numero1, numero2, operacao)}");

operacao = (a, b) => a - b;

Console.WriteLine($"O resultado da operação é {Operar(numero1, numero2, operacao)}");

Console.ReadKey();
static int Operar(int num1, int num2, Func<int, int, int> operacao)
{
    return operacao(num1, num2);
}
```

Nesse exemplo, o método Main define duas variáveis inteiras numero1 e numero2. Em seguida, ele define um delegate Func<int, int, int> chamado operacao que recebe dois inteiros como parâmetros e retorna um inteiro. Na primeira execução, o delegate é configurado para realizar a operação de adição. Na segunda execução, o delegate é configurado para realizar a operação de subtração.

O método Main chama o método Operar passando as duas variáveis inteiras e o delegate operacao como parâmetros, e imprime o resultado retornado por Operar na tela.

O método Operar recebe dois números inteiros e um delegate Func<int, int, int> como parâmetros, e retorna o resultado da operação definida pelo delegate. No exemplo acima, o método Operar retorna a soma ou a subtração dos números passados como parâmetros, dependendo da operação definida pelo delegate.

7- Suponha que você possui uma lista de planetas com informações sobre o nome, diâmetro e massa de cada planeta.

Delegates , Lambda, LINQ

Crie um programa e implemente um método que recebe essa lista e um delegate Predicate<Planeta> para filtrar os planetas de acordo com um critério específico.
Resposta:

```
List<Planeta> planetas = new List<Planeta>
{
    new Planeta("Mercúrio", 4879, 3.301e23),
            new Planeta("Vênus", 12104, 4.867e24),
            new Planeta("Terra", 12756, 5.972e24),
            new Planeta("Marte", 6792, 6.39e23),
            new Planeta("Júpiter", 142984, 1.898e27),
            new Planeta("Saturno", 120536, 5.683e26),
            new Planeta("Urano", 51118, 8.681e25),
            new Planeta("Netuno", 49528, 1.024e26)
};

// Filtrar os planetas com diâmetro maior que 10000 km
Predicate<Planeta> filtro = p => p.Diametro > 10000;
List<Planeta> planetasGrandes = Filtrar(planetas, filtro);

Console.WriteLine("Planetas com diâmetro maior que 10000 km:");
foreach (Planeta planeta in planetasGrandes)
{
    Console.WriteLine(planeta.Nome);
}
Console.ReadKey();

static List<Planeta> Filtrar(List<Planeta> lista, Predicate<Planeta> filtro)
{
    List<Planeta> resultado = new List<Planeta>();
    foreach (Planeta planeta in lista)
    {
        if (filtro(planeta))
        {
            resultado.Add(planeta);
        }
    }
    return resultado;
}

class Planeta
{
    public string Nome { get; set; }
    public double Diametro { get; set; }
    public double Massa { get; set; }
    public Planeta(string nome, double diametro, double massa)
    {
        Nome = nome;
        Diametro = diametro;
        Massa = massa;
    }
}
```

Nesse exemplo, o método Main cria uma lista de planetas e define um delegate Predicate<Planeta> chamado filtro que verifica se o diâmetro de um planeta é maior que 10000 km. O método Main chama o método Filtrar passando a lista de planetas e o delegate filtro como parâmetros, e armazena o resultado em uma nova lista chamada planetasGrandes.

O método Filtrar recebe uma lista de planetas e um delegate Predicate<Planeta> como parâmetros, e retorna uma nova lista contendo apenas os planetas da lista original que satisfazem o predicado definido pelo delegate. No exemplo acima, o método Filtrar retorna uma lista contendo apenas os planetas com diâmetro maior que 10000 km.

Por fim, o método Main imprime os nomes dos planetas retornados pelo método Filtrar na tela.

8 - Qual é a definição correta de expressão lambda em C#?

a) Uma função anônima que pode ser passada como argumento para outros métodos e que pode ser definida em apenas uma linha de código.
b) Uma função que sempre retorna um valor booleano.
c) Uma função que sempre tem um único parâmetro de entrada.
d) Uma função que sempre retorna um valor inteiro.

Resposta: a) Uma função anônima que pode ser passada como argumento para outros métodos e que pode ser definida em apenas uma linha de código.

Justificativa: A definição de expressão lambda em C# é uma função anônima que pode ser passada como argumento para outros métodos. É uma maneira concisa e elegante de escrever funções que geralmente têm apenas uma linha de código. Elas podem ter qualquer número de parâmetros e retornar qualquer tipo de valor. É uma das características mais poderosas da linguagem C#.

9- Crie um método de extensão para a classe List<int> na linguagem C# que, dado uma lista de inteiros, retorne a soma de todos os elementos da lista que são ímpares.

Em seguida, crie um programa que instancie uma lista de inteiros com alguns números ímpares e alguns números pares e use o método de extensão criado para obter a soma dos números ímpares da lista.

Resposta

```
List<int> numeros = new List<int>() { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

int soma = numeros.SomarNumerosImpares();

Console.WriteLine("Soma de números ínpares " + soma);

Console.ReadKey();

public static class ListExtensions
{
    public static int SomarNumerosImpares(this List<int> list)
    {
        int soma = 0;
        foreach (int n in list)
        {
            if (n % 2 == 1)
            {
                soma += n;
            }
        }
```

```
        return soma;
    }
}
```

A implementação cria um método de extensão chamado SomarNumerosImpares para a classe List<int>, que permite que a lista de inteiros possa chamar o método diretamente.

O método itera sobre todos os elementos da lista e verifica se cada um é ímpar. Se for, o número é adicionado à variável sum, que é a soma de todos os números ímpares da lista. Por fim, o método retorna o valor de sum.

No programa principal, uma lista de inteiros é criada com alguns números pares e ímpares. Em seguida, o método de extensão SomarNumerosImpares é chamado para calcular a soma dos números ímpares da lista. O resultado é impresso na tela usando o método Console.WriteLine.

O uso do método de extensão torna o código mais legível e reutilizável, pois permite que o método SomarNumerosImpares seja usado em qualquer lista de inteiros, sem a necessidade de criar um novo método de soma para cada lista que desejarmos calcular a soma dos números ímpares.

10 - Exercicios sobre LINQ

Exercício 1: Utilizando o método Where

Dado um array de strings, escreva um programa em C# que use a biblioteca LINQ para obter somente as strings que contenham a letra 'a'.
Resposta:

```
string[] frutas = { "banana", "abacaxi", "uva", "laranja", "abacate", "Kiwi" };

var result = frutas.Where(w => w.Contains('a'));

Console.WriteLine("\nFrutas que contém a letra 'a':");
foreach (var fruta in result)
{
    Console.Write(" " + fruta + " ");
}
Console.ReadKey();
```

Exercício 2: Utilizando o método OrderBy

Dado um array de inteiros, escreva um programa em C# que use a biblioteca LINQ para ordenar os valores em ordem crescente.
Resposta:

```
int[] numeros = { 5, 2, 8, 3, 1, 7, 4, 6 };
var result = numeros.OrderBy(n => n);

Console.WriteLine("Números na ordem ascendente:");
foreach (var n in result)
{
    Console.Write(n + " ");
}

Console.ReadKey();
```

Exercício 3: Utilizando o método GroupBy

Delegates , Lambda, LINQ

Dado um array de strings, escreva um programa em C# que use a biblioteca LINQ para agrupar as strings por tamanho
Resposta:

```
string[] palavras = { "banana", "abacaxi", "uva","Kiwi", "laranja", "abacate",
"maça", "pera" };

var result = palavras.GroupBy(p => p.Length);

Console.WriteLine("Palavras agrupadas por tamanho:");

foreach (var group in result)
{
    Console.WriteLine("\nPalavras com tamanho " + group.Key + ":");
    foreach (var palavra in group)
    {
        Console.Write(" " + palavra + " ");
    }
    Console.WriteLine();
}
Console.ReadKey();
```

Exercício 4: Utilizando o método FirstOrDefault

Dado um array de inteiros, escreva um programa em C# que use a biblioteca LINQ para obter o primeiro número par do array.
Resposta:

```
int[] numeros = { 5, 2, 8, 3, 1, 7, 4, 6 };

int result = numeros.FirstOrDefault(n => n % 2 == 0);

Console.WriteLine("Primeiro número par : " + result);

Console.ReadKey();
```