Generics e Coleções Genéricas – Exercícios

1- Quais das seguintes classes estão presentes no nameaspace System.Collections.Generic :
Stack<T> 2-Tree<T> 3-SortedDictionary<T> 4-SortedArray<T>

a-) 1 e 2 somente
b-) 2 e 4 somente
c-) 1 e 3 somente
d-) Todas as opções
e-) Nenhuma das opções

Resposta : c
Somente Stack<T> e SortedDictionary<T> estão neste namespace

2- Dado o trecho de código abaixo :

```
public class Generic<T>
{
   public T Campo;
   public void TesteSub()
   {
      T i = Campo + 1;
   }
}

class MeuPrograma
{
   static void Main(string[] args)
   {
      Generic<int> gen = new Generic<int>();
      gen.TesteSub();
   }
}
```

Dado o trecho de código acima , qual das seguintes declarações são verdadeiras ?

a-) A adição produzirá o resultado 1.
b-) O resultado da adição depende do sistema.
c-) O programa gerará uma exceção em tempo de execução.
d-) O compilador vai relatar o erro: O operador '+' não está definido para os tipos T e int.
e-) Nenhuma das acima.

Resposta:

O compilador relatará um erro**: O operador '+' não está definido para os tipos T e int.**

Este código define uma classe genérica Generic com um parâmetro de tipo genérico T, que é usado para declarar um campo Campo do tipo T.

Generics e Coleções Genéricas - Exercícios

A classe também possui um método TesteSub que cria uma variável local i e atribui seu valor como resultado de Campo + 1.

No método Main da classe MeuProgram, uma instância da classe Generic<int> é criada e armazenada na variável gen. O método TesteSub é então chamado na instância gen.

No entanto, como o operador de adição (+) não está definido para o tipo genérico T, esse código não será compilado e resultará em um erro de tempo de compilação. Para resolver o problema, a classe ou método que usa T precisa ser restringido para permitir apenas tipos para os quais o operador de adição é definido.

3- Quais das declarações a seguir são verdadeiras para o recurso Generics da plataforma .NET ?

1- Generics é um recurso de linguagem.
2- Podemos criar uma classe genérica, porém não podemos criar uma interface genérica em C#
3- Delegates genéricos não são permitidos em C#.
4- O recurso Generics são úteis em classes de coleção na plataforma .NET

a- 1 e 2 somente
b- 1, 2 e 3 somente
c- 1 e 4 somente
d- Todas as opções
e- Nenhuma das opções

Resposta : d

4- Qual o resultado da execução do código abaixo:

```
Teste teste = new Teste();
teste.MetodoTeste<string>("Usando Generics -> ");
teste.MetodoTeste<float>(4.2f);

Console.ReadKey();

public class Teste
{
    public void MetodoTeste<T>(T arg)
    {
        Console.Write(arg);
    }
}
```

a-) O programa vai compilar e na execução imprimirá: Generics -> 4.2
b-) Uma classe não genérica Teste não pode ter um método genérico MetodoTeste<T>
c-) O compilador vai gerar um erro.
d-) O programa irá gerar uma exceção em tempo de execução
e-) Nenhuma das opções acima.

Resposta: a

Generics e Coleções Genéricas - Exercícios

Uma classe não genérica pode ter um método genérico.
Poderíamos ter omitido o tipo na chamada do método:

```
teste.MetodoTeste("Usando Generics -> ");
teste.MetodoTeste(4.2f);
```

O compilador vai inferir o tipo padrão usado pelo argumento informado.

5- Qual o resultado da execução do código abaixo:

```
Generic<String> g = new Generic<String>();
g.Campo = "Exercício Generics";
Console.WriteLine(g.Campo);

Console.ReadKey();

public class Generic<T>
{
    public T? Campo;
}
```

a-) O nome Generic não pode ser usado como um nome de classe porque é uma palavra-chave.
b-) Não podemos criar uma classe genérica e definir um campo genérico
c-) Vai imprimir a string "**Exercício Generics**" no console.
d-) O campo de membro da classe Generic não é acessível diretamente.
e-) Nenhuma das acima.

Resposta: c

Este código é um exemplo de uso de Generics em C#. Ele cria uma instância da classe "Generic" com o tipo genérico sendo "String".
O campo "Campo" é então definido como "Exercício Generics" e impresso na tela usando a instrução Console.WriteLine().
A saída deste código será: "Exercício Generics".
Observe que o tipo genérico "T" é declarado como "Nullable", o que significa que o campo "Campo" pode ser nulo, mesmo que o tipo "String" não seja nullable por padrão.

6- Para o trecho de código fornecido abaixo, quais das seguintes declarações é válida ?

```
public class MeuContainer<T> where T: class, IComparable
{
    // Insira o código aqui
}
```

a-) A classe MeuContainer requer que seu tipo de argumento implemente a interface IComparable.
b-) O argumento do tipo da classe MeuContainer deve ser IComparable.
c-) O compilador reportará um erro para este bloco de código.
d-) A classe MeuContainer requer que seu argumento de tipo seja um tipo de referência e implemente a interface IComparable.

Generics e Coleções Genéricas - Exercícios

Resposta: d

Este é um exemplo de uso de restrições de tipo genérico em C#. A classe "MeuContainer" é uma classe genérica que permite que o tipo genérico "T" seja qualquer tipo que seja uma classe e que implemente a interface "IComparable".

A palavra-chave "where" é usada para impor uma restrição de tipo genérico. Nesse caso, a restrição "where T: class, IComparable" especifica que o tipo genérico "T" deve ser uma classe e também deve implementar a interface "IComparable".

Isso significa que, ao instanciar a classe "MeuContainer", você só pode usar tipos que sejam classes e que implementem a interface "IComparable", como tipos numéricos (int, double, etc.), tipos de data e hora (DateTime, TimeSpan, etc.) e outros tipos que sejam classes e implementem "IComparable".

O código dentro da classe "MeuContainer" pode então aproveitar as funcionalidades da interface "IComparable" para comparações entre objetos do tipo "T".

7- Qual das seguintes afirmações é válida sobre as vantagens dos genéricos?

a-) Generics transferem o ônus da segurança de tipo para o programador em vez do compilador.
b-) Generics requerem o uso de conversão de tipo explícito.
c-) Generics fornecem segurança de tipo sem a sobrecarga de várias implementações.
d-) Generics eliminam a possibilidade de erros de execução.
e-) Nenhuma das acima

Resposta: c

Os genéricos permitem que você escreva código genérico que possa ser usado com vários tipos diferentes, sem precisar escrever várias implementações para cada tipo.
Isso resulta em código mais limpo, fácil de manter e com menos erros.

Embora os genéricos ofereçam segurança de tipo, ainda é importante que o programador escreva código seguro e evite erros de programação. Eles não eliminam completamente a possibilidade de erros de execução.

As outras afirmações não são verdadeiras.

Os genéricos não transferem o ônus da segurança de tipo para o programador, eles oferecem segurança de tipo através da verificação do tipo durante a compilação.

Eles também não requerem o uso de conversão de tipo explícito, pois o compilador resolve o tipo a ser usado automaticamente.

Generics e Coleções Genéricas - Exercícios

8- Escreva um programa para adicionar dois números inteiros usando o conceito de Generics.

Resposta:

```
AdicionaNumeros<int> adicionaInteiros = new AdicionaNumeros<int>();

Console.WriteLine("Informe o primeiro número :");
var num1 = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("Informe o segundo número :");
var num2 = Convert.ToInt32(Console.ReadLine());

var resultado = adicionaInteiros.Adiciona(num1, num2);

Console.WriteLine("Resultado da adição: " + resultado);

Console.ReadKey();

public class AdicionaNumeros<T> where T : struct, IComparable, IConvertible,
IFormattable
{
    public T Adiciona(T num1, T num2)
    {
        dynamic dynamicNum1 = num1;
        dynamic dynamicNum2 = num2;
        return dynamicNum1 + dynamicNum2;
    }
}
```

Nesse exemplo, a classe AdicionaNumeros é uma classe genérica que permite que o tipo genérico T seja qualquer tipo numérico estruturado que implemente as interfaces IComparable, IConvertible e IFormattable. A restrição de tipo where T : struct, IComparable, IConvertible, IFormattable é especificada para garantir que o tipo genérico T seja compatível com operações matemáticas.

A classe AdicionaNumeros possui um método Adiciona que adiciona dois números do tipo T e retorna o resultado. O uso de dynamic permite que o compilador resolva o tipo do número em tempo de execução, permitindo que o método seja usado com qualquer tipo numérico estruturado.

No método Main, uma instância da classe AdicionaNumeros é criada para números inteiros, e os números 10 e 20 são adicionados usando o método Adiciona. O resultado é exibido na tela.

Generics e Coleções Genéricas - Exercícios

9- Escreva um programa que crie uma lista de objetos Aluno que contém as propriedades : Nome, Idade e Sexo

A seguir defina 5 objetos do tipo Aluno e exiba uma lista de objetos alunos no console.

Resposta:

```
List<Aluno> listaAlunos = new List<Aluno>();

listaAlunos.Add(new Aluno("João", 22, 'M'));
listaAlunos.Add(new Aluno("Maria", 21, 'F'));
listaAlunos.Add(new Aluno("Pedro", 23, 'M'));
listaAlunos.Add(new Aluno("Ana", 20, 'F'));
listaAlunos.Add(new Aluno("Lucas", 24, 'M'));

Console.WriteLine("Lista de Alunos:\n");

foreach (Aluno aluno in listaAlunos)
{
    Console.WriteLine($"Nome: {aluno.Nome} Idade: {aluno.Idade} Sexo: {aluno.Sexo}");
}

Console.ReadKey();
public class Aluno
{
    public string Nome { get; set; }
    public int Idade { get; set; }
    public char Sexo { get; set; }

    public Aluno(string nome, int idade, char sexo)
    {
        Nome = nome;
        Idade = idade;
        Sexo = sexo;
    }
}
```

10- implemente um programa que verifica se uma expressão matemática contém parênteses balanceados seguindo os seguintes passos:

1. Crie uma variável do tipo Stack<char> para armazenar os parênteses abertos.
2. Percorra cada caractere da expressão matemática.
3. Se o caractere for um parêntese aberto ('(', '{', '['), adicione-o à pilha.
4. Se o caractere for um parêntese fechado (')', '}', ']'), verifique se a pilha não está vazia e se o último parêntese aberto adicionado na pilha corresponde ao parêntese fechado atual. Se sim, remova o último parêntese aberto da pilha. Caso contrário, a expressão matemática não contém parênteses balanceados.
5. Após percorrer todos os caracteres da expressão matemática, verifique se a pilha está vazia. Se estiver vazia, a expressão matemática contém parênteses balanceados. Caso contrário, a expressão não é balanceada.

Generics e Coleções Genéricas - Exercícios

Resposta: Abaixo está um exemplo de implementação do exercício proposto:

```
string expression = "(1 + 2) * 3) - 4";

bool balenceada = EstaBalanceada(expression);

Console.WriteLine($"A expressão matemática {expression} é {(balenceada ?
            "esta balanceada" : "não esta balanceada")}.");

Console.ReadKey();

static bool EstaBalanceada(string expression)
{
   Stack<char> stack = new Stack<char>();

   foreach (char c in expression)
   {
      if (c == '(' || c == '{' || c == '[')
      {
         stack.Push(c);
      }
      else if (c == ')' || c == '}' || c == ']')
      {
         if (stack.Count == 0)
         {
            return false;
         }

         char lastParentheses = stack.Pop();

         if ((c == ')' && lastParentheses != '(') ||
           (c == '}' && lastParentheses != '{') ||
           (c == ']' && lastParentheses != '['))
         {
            return false;
         }
      }
   }

   return stack.Count == 0;
}
```

Neste código a expressão matemática "(1 + 2) * 3) - 4" não é balanceada, pois contém um parêntese fechado ')' que não possui um parêntese aberto correspondente na pilha.

Generics e Coleções Genéricas - Exercícios

11. Implementar um programa que simula uma fila de impressão seguindo o seguinte roteiro:

- Crie uma variável do tipo Queue<string> para representar a fila de impressão.
- Crie um loop que irá executar até que a fila de impressão esteja vazia.
- Dentro do loop, verifique se a fila de impressão não está vazia. Se não estiver vazia, remova o primeiro elemento da fila usando o método Dequeue() e imprima na tela que o arquivo "X" está sendo impresso.
- Simule o tempo de impressão com um Thread.Sleep() por um período aleatório de tempo entre 1 e 5 segundos.
- Após simular a impressão do arquivo, imprima na tela que o arquivo "X" foi impresso com sucesso.
- Repita os passos 3 a 5 até que a fila de impressão esteja vazia.

Resposta:

```
Queue<string> printQueue = new Queue<string>();

printQueue.Enqueue("Arquivo1.pdf");
printQueue.Enqueue("Arquivo2.docx");
printQueue.Enqueue("Arquivo3.ppt");

while (printQueue.Count > 0)
{
    string file = printQueue.Dequeue();

    Console.WriteLine($"\nO arquivo \"{file}\" está sendo impresso...");

    Thread.Sleep(new Random().Next(1000, 5000));

    Console.WriteLine($"O arquivo \"{file}\" foi impresso com sucesso.");
}

Console.ReadKey();
```

12 - Escreva um programa seguindo as seguintes orientações:

Declare um método genérico chamado **ReverterEImprimir** em uma classe não genérica chamada Exemplo.
O método recebe como parâmetro um array de qualquer tipo.
A seguir declare três tipos diferentes de array : um array de int , um array de strings e um array de double
Invoque o método duas vezes com cada array.
Na primeira vez invoque o método com um determinado array, onde ele usa explicitamente o parâmetro de tipo.
Na segunda vez, invoque o método onde o tipo é inferido.

Resposta:

Generics e Coleções Genéricas - Exercícios

```
int[] intArray = { 1, 2, 3, 4, 5 };
string[] stringArray = { "Lucas", "Rafael", "Larissa", "João" };
double[] doubleArray = { 1.5, 2.5, 3.5, 4.5, 5.5 };

Console.WriteLine("-Array de Inteiros:");
Exemplo.ReverterEImprimir<int>(intArray);
Exemplo.ReverterEImprimir(intArray);

Console.WriteLine("\n -Array de Strings:");
Exemplo.ReverterEImprimir<string>(stringArray);
Exemplo.ReverterEImprimir(stringArray);

Console.WriteLine("\n-Array de Doubles:");
Exemplo.ReverterEImprimir<double>(doubleArray);
Exemplo.ReverterEImprimir(doubleArray);


Console.ReadKey();
class Exemplo
{
    public static void ReverterEImprimir<T>(T[] array)
    {
        Array.Reverse(array);
        Console.WriteLine("Array Revertido:");
        foreach (T item in array)
        {
            Console.Write(item + " ");
        }
        Console.WriteLine();
    }

}
```