

Tratamento de erros

1- Defina como verdadeiro ou falso as declarações a seguir

Uma exceção é uma indicação de que ocorreu um erro durante a execução de um programa. (Verdadeiro)

A classe System.Exception é a classe base de todas as exceções em C#. (Verdadeiro)

A pilha de execução (stack trace) mostra a sequência de chamadas de métodos que levaram à exceção. (Verdadeiro)

As exceções em C# são organizadas em uma hierarquia de classes, com a classe System.Exception no topo da hierarquia. (Verdadeiro)

Todas as exceções em C# são subclasses da classe System.Exception. (Verdadeiro)

2- Defina como verdadeiro ou falso as declarações a seguir

O bloco try é usado para envolver o código que pode gerar uma exceção. (Verdadeiro)

O bloco catch é usado para capturar a exceção e lidar com ela de alguma forma. (Verdadeiro)

O bloco finally é opcional e é usado para executar código, independentemente de ter ocorrido ou não uma exceção. (Verdadeiro)

O bloco finally é sempre executado antes do bloco catch. (Falso)

É possível ter vários blocos catch para capturar diferentes tipos de exceções. (Verdadeiro)

3 - Escreva um program onde o usuário é solicitado a informar um valor via teclado e armazenar o valor na variável entrada do tipo string onde tem que verificar 3 condições diferentes:

a- Se a entrada é nula, uma exceção NullReferenceException é lançada com a mensagem "A entrada não pode ser nula."

b-Se a entrada está vazia, uma exceção ArgumentException é lançada com a mensagem "A entrada não pode estar vazia."

c-Se a entrada passar nas verificações anteriores, a entrada é exibida na tela.

Realize o tratamento de erro para essas condições usando o bloco try-cath-finally

Resposta:

```
Try
{
    string input = Console.ReadLine();
    if (input == null)
    {
        throw new NullReferenceException("A entrada não pode ser nula.");
    }
    if (input.Length == 0)
    {
        throw new ArgumentException("A entrada não pode estar vazia.");
    }
    Console.WriteLine("A entrada é: " + input);
}
catch (NullReferenceException e)
{
}
```

Tratamento de erros

```
    Console.WriteLine("Erro: " + e.Message);  
}  
catch (ArgumentException e)  
{  
    Console.WriteLine("Erro: " + e.Message);  
}  
catch (Exception e)  
{  
    Console.WriteLine("Erro inesperado: " + e.Message);  
}  
finally  
{  
    Console.WriteLine("O programa foi concluído.");  
}
```

O bloco try-catch é usado para capturar exceções e tratar cada uma delas de forma diferente. Se uma exceção `NullReferenceException` ou `ArgumentException` for lançada, uma mensagem de erro será exibida na tela com a mensagem apropriada. Se outra exceção for lançada, uma mensagem de erro mais geral será exibida.

O bloco finally é usado para executar código que deve ser executado independentemente se ocorreu uma exceção ou não. Neste caso, uma mensagem é exibida na tela para indicar que o programa foi concluído.

Para testar este exercício, tente inserir diferentes tipos de entrada, como uma entrada nula, uma entrada vazia ou uma entrada com conteúdo válido. Veja como o programa trata cada situação e se a mensagem apropriada é exibida na tela.

4- Escreva um programa que solicite ao usuário a informação da idade e do nome via teclado que deverão ser armazenados nas variáveis idade do tipo int e nome do tipo string.

A seguir realize o tratamento de erro e lançando as exceções considerando as seguintes condições:

a- Se a idade é negativa, uma exceção `ArgumentException` é lançada com a mensagem "A idade não pode ser negativa."

b - Se a idade é zero, uma exceção `NotImplementedException` é lançada com a mensagem "A idade ainda não foi definida."

c- Se o nome é nulo, uma exceção `NullReferenceException` é lançada com a mensagem "O nome não pode ser nulo."

Resposta:

```
int idade = 0;  
string nome = null;  
try  
{  
    Console.WriteLine("Informe a sua idade");  
    idade = Convert.ToInt32(Console.ReadLine());  
    if (idade < 0)  
    {
```

Tratamento de erros

```
        throw new ArgumentException("A idade não pode ser negativa.");
    }
    if (idade == 0)
    {
        throw new NotImplementedException("A idade ainda não foi definida.");
    }

    Console.WriteLine("Informe o seu nome");
    nome = Console.ReadLine();

    if (nome == null)
    {
        throw new NullReferenceException("O nome não pode ser nulo.");
    }
}
catch (ArgumentException e)
{
    Console.WriteLine("Erro de argumento: " + e.Message);
}
catch (NotImplementedException e)
{
    Console.WriteLine("Erro de implementação: " + e.Message);
}
catch (NullReferenceException e)
{
    Console.WriteLine("Erro de referência nula: " + e.Message);
}
catch (Exception e)
{
    Console.WriteLine("Erro inesperado: " + e.Message);
}
```

Tratamento de erros

Cada exceção é capturada por um bloco catch correspondente. Se uma exceção `ArgumentException`, `NotSupportedException` ou `NullReferenceException` for lançada, uma mensagem de erro será exibida na tela com a mensagem apropriada. Se outra exceção for lançada, uma mensagem de erro mais geral será exibida.

Para testar este exercício, tente mudar o valor da variável `idade` para um valor válido, defina o valor da variável `nome` para um valor não-nulo ou adicione mais exceções personalizadas para serem lançadas. Veja como o programa trata cada situação e se a mensagem apropriada é exibida na tela.

5- Dado um array de inteiros expresso da seguinte forma

```
int[] numeros = new int[] { 109, 211, 313, 405, 519, 617, 711, 891, 951, 1001 };
```

Exiba na janela do console os numeros do array e solicite via teclado ao usuário para informar o valor de um índice do array para obter o seu respectivo valor.

Realize o tratamento de exceções filtrando as exceções `IndexOutOfRangeException` e `ArgumentNullException`

Resposta:

```
try
{
    int[] numeros = new int[] { 109, 211, 313, 405, 519, 617, 711, 891, 951, 1001 };
    int index = 99;
    Console.WriteLine("Dado um array de números inteiros contendo os seguintes valores:");
    for (int i = 0; i < numeros.Length; i++)
    {
        Console.Write(numeros[i] + " ");
    }
    Console.WriteLine("\n\nInforme o índice do array para obter um valor: ");
    index= Convert.ToInt32(Console.ReadLine());
    Console.WriteLine("\nO número na posição " + index + " é " + numeros[index]);
}
catch (Exception e) when (e is IndexOutOfRangeException || e is ArgumentNullException)
{
    Console.WriteLine("\nErro: " + e.Message);
}
Console.ReadKey();
```

Tratamento de erros

Uma cláusula try-catch é usada para capturar exceções que podem ser lançadas por essa instrução.

Uma cláusula when é adicionada à cláusula catch para filtrar as exceções que serão capturadas.

Neste caso, estamos interessados em capturar exceções do tipo `IndexOutOfRangeException` ou `ArgumentNullException`, que podem ser lançadas quando o índice é inválido ou o array é nulo, respectivamente.

Se uma exceção `IndexOutOfRangeException` ou `ArgumentNullException` for lançada, a cláusula when capturará a exceção e exibirá uma mensagem de erro na tela. Se outra exceção for lançada, ela não será capturada e o programa encerrará.

Para testar este exercício, tente alterar o valor do índice `index` para um valor dentro dos limites do array, defina o array como nulo ou adicione mais filtros de exceção para serem capturados pela cláusula when.

Veja como o programa trata cada situação e se a mensagem apropriada é exibida na tela.

6- Dado o seguinte código:

```
try
{
    int saldo = 0;
    int valorSaque = 100;
    if (valorSaque > saldo)
    {
        throw new SaldoInsuficienteException("O saldo é insuficiente para este saque.");
    }
    saldo -= valorSaque;
    Console.WriteLine("Saque efetuado com sucesso. Novo saldo: " + saldo);
}
catch (SaldoInsuficienteException e)
{
    Console.WriteLine("Erro: " + e.Message);
}
```

Implemente a exceção personalizada `SaldoInsuficienteException`.

Resposta:

```
class SaldoInsuficienteException : Exception
{
    public SaldoInsuficienteException() { }
    public SaldoInsuficienteException(string message) : base(message) { }
    public SaldoInsuficienteException(string message, Exception innerException) : base(message, innerException) { }
}
```

Tratamento de erros

Neste exercício, uma variável saldo é inicializada com um valor de 0, representando o saldo de uma conta bancária. Em seguida, uma variável valorSaque é definida com um valor de 100, que é maior que o saldo atual.

Uma verificação é realizada para verificar se o valor do saque é maior que o saldo. Se for, uma exceção personalizada SaldoInsuficienteException é lançada com a mensagem "O saldo é insuficiente para este saque."

Uma cláusula try-catch é usada para capturar essa exceção personalizada e exibir uma mensagem de erro na tela.

A exceção personalizada SaldoInsuficienteException é definida como uma classe separada que herda da classe base Exception. Ela contém três construtores: um construtor sem argumentos, um construtor que recebe uma mensagem como argumento e um construtor que recebe uma mensagem e uma exceção interna como argumentos.

Para testar este exercício, tente alterar o valor da variável valorSaque para um valor menor que o saldo, adicione mais verificações ou exceções personalizadas, ou defina a variável saldo como nula. Veja como o programa trata cada situação e se a mensagem apropriada é exibida na tela.

7- Considere o seguinte código C#:

```
static void MeuMetodo(int valor)
{
    try
    {
        if (valor < 0)
        {
            throw new MinhaException("Valor negativo não permitido.");
        }
        else if (valor > 100)
        {
            throw new ArgumentException("O valor não pode ser maior que 100.");
        }
        Console.WriteLine("O valor é válido.");
    }
    catch (MinhaException e) when (valor < 0)
    {
        Console.WriteLine("Erro: " + e.Message);
    }
    catch (ArgumentException e) when (valor > 100)
    {
        Console.WriteLine("Erro: " + e.Message);
    }
}
```

Tratamento de erros

```
}  
catch (Exception e)  
{  
    Console.WriteLine("Erro genérico: " + e.Message);  
}  
finally  
{  
    Console.WriteLine("Método concluído.");  
}  
}  
public class MinhaException : Exception  
{  
    public MinhaException() { }  
    public MinhaException(string message) : base(message) { }  
}
```

Qual é o resultado esperado da execução do método MeuMetodo(-5)?

- A) A mensagem "Erro genérico: Valor negativo não permitido." é exibida na tela.
- B) A mensagem "Erro genérico: O valor não pode ser maior que 100." é exibida na tela.
- C) A mensagem "Erro: Valor negativo não permitido." é exibida na tela.
- D) A mensagem "Erro: O valor não pode ser maior que 100." é exibida na tela.
- E) Nenhuma mensagem de erro é exibida na tela.

Resposta correta: C.

A condição when no primeiro bloco catch garante que a exceção personalizada MinhaException só será tratada se o valor passado para o método for negativo.

Como o valor passado é -5, essa condição é verdadeira e a mensagem "Erro: Valor negativo não permitido." é exibida na tela.

Os outros blocos catch são ignorados porque a exceção lançada não corresponde às condições definidas.

O bloco finally sempre é executado, então a mensagem "Método concluído." também é exibida na tela.