cencal centro de formação profissional para a indústria de cerâmica

Programação em C#

António Ramos | Outubro 2022

Co-Financiado por:

IEFP APICER associação portuguesa das indústrias de cerâmica e cristalaria ARO Associação Empresarial da Região Oeste REPÚBLICA PORTUGUESA POCH PORTUGAL 2020 UNIÃO EUROPEIA Fundo Social Europeu

Operadores de comparação

A estrutura de seleção condicional envolve a utilização de declarações condicionais.

O programa irá diferenciar se certa condição é (ou não) correspondida.

Operadores de comparação

==

É igual

Operadores de comparação

!=

Não é igual / É diferente

Operadores de comparação

>

Maior que

Operadores de comparação

<

Menor que

Operadores de comparação

>=

Maior ou Igual

Operadores de comparação

<=

Menor ou Igual

Operadores de comparação

&&

(E) Combinar múltiplas condições

Operadores de comparação

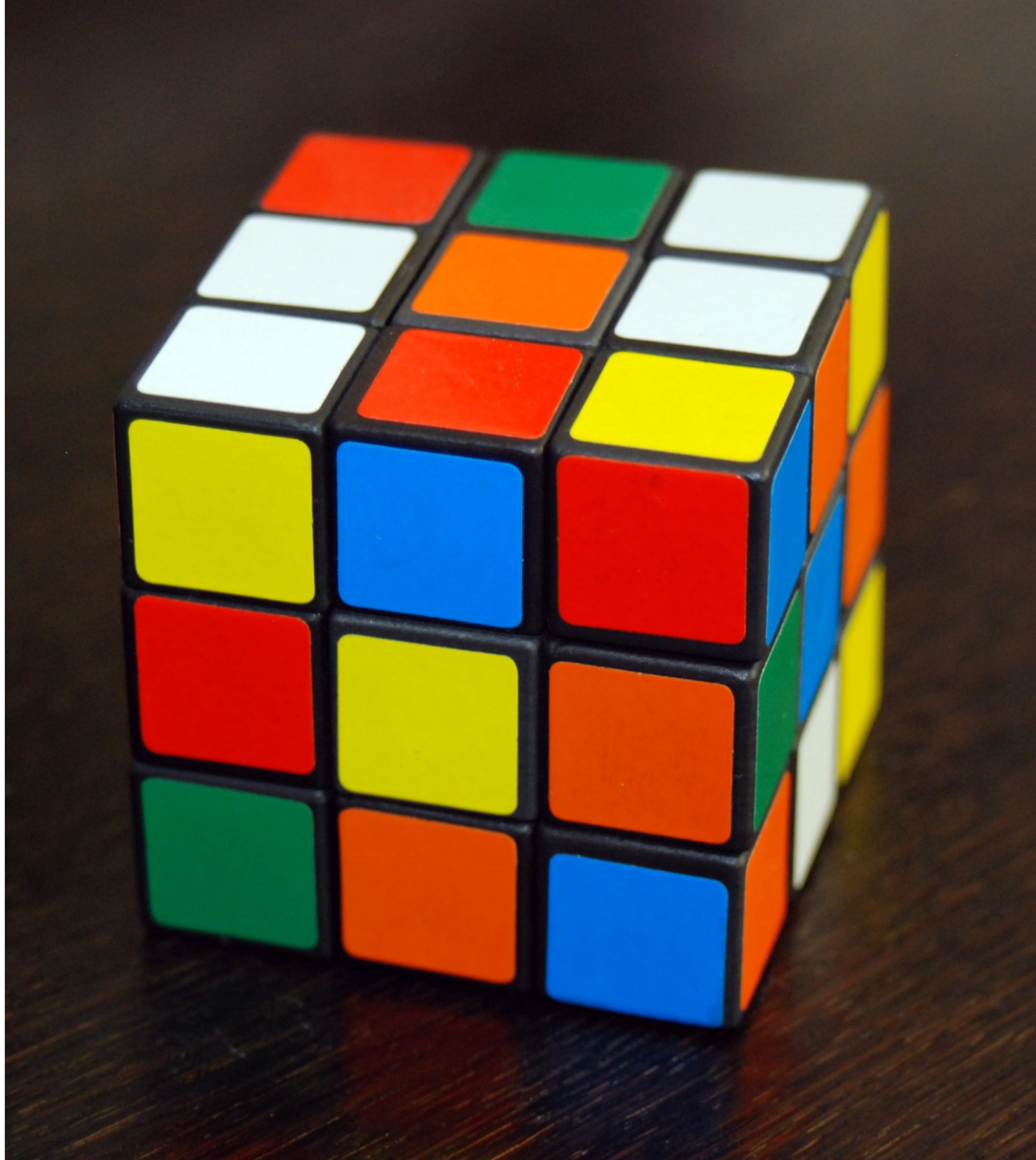
||

(OU) Combinar múltiplas condições

cencal centro de formação profissional para a indústria de cerâmica

Instruções de Controlo de Fluxo

Em conjunto com as operadores de comparação, permite adicionar lógica e controlar o fluxo do nosso programa.



Instrução IF

Permite ao programa avaliar se certa condição é correspondida, sendo posteriormente efetuada a ação a que esta corresponde.

```
if (condition){

}
```

```
if (condition){

}else if(condition2){

}else {

}
```

Instrução IF

Permite ao programa avaliar se certa condição é correspondida, sendo posteriormente efetuada a ação a que esta corresponde.

```
variable = (condition) ? expressionTrue :  expressionFalse;
```

Instrução Switch

Comparação é directa a um único valor, e não uma variedade de condições.

O caso "default" é opcional e apenas corre se nenhuma das condições anteriores for correspondida.

A keyword "break" é usada para o compilador saltar a condição.

```
switch(expression)
{
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
    break;
}
```

Ciclos (Loops)

Permitem executar determinado bloco de código até ser atingida uma condição.

É útil porque ajuda na optimização do código e do tempo, reduzindo erros e tornando o código mais fácil de ler.

Ciclo For

Executa um bloco de código até que a condição seja atingida.

É utilizado quando sabemos o número de vezes que queremos correr a condição.

```
for (int i = 0; i < 5; i++)
{
   Console.WriteLine(i);
}
```

- int i = 0
  - Declaração e inicialização da variável "contador"
- i < 5
  - Enquanto condição for positiva, corre o bloco de código
- i++
  - No fim do ciclo, incrementa o contador

Ciclo Foreach

Usado exclusivamente para iterar elementos de um array ou lista.

Vantajoso quando pretendemos obter informação, sem efetuar alterações.

```
foreach (type variableName in arrayName)
{
  // code block to be executed
}
```

Ciclo While

Atenção: Pode correr infinitamente.

Necessita de variável que sirva de condição / contador, sendo reduzida até que a condição cumpra dentro do ciclo.

```
while (condition){
  // code block to be executed
}
```

Ciclo Do

Repete o bloco de código até que a condição seja cumprida

```
do{
  // code block to be executed
}
while (condition);
```

Instruções de Salto

Indicam ao programa para desviar do fluxo normal de sequência e saltar para outro bloco de código.

Break

Return

Continue

Throw

Instruções de Salto

Break

Pode ser utilizada em ciclos e faz sair do Loop quando certa condição é correspondida.

```
int[] numbers = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
foreach (int number in numbers)
{
    if (number == 3)
    {
        break;
    }

    Console.Write($"{number} ");
}
```

Instruções de Salto

Continue

Pode ser utilizada em ciclos e faz sair do Loop quando certa condição é correspondida.

```
for (int i = 0;  i<5;  i++)
{
    if (i == 2) continue;
    Console.WriteLine("i = {0}", i);
}
```

Instruções de Salto

Return

Termina a execução da função e retorna o controlo ou resultado

```
DisplayIfNecessary(6);

void DisplayIfNecessary(int number)
{
    if (number % 2 == 0)
    {
        return;
    }
    Console.WriteLine(number);
}
```

Instruções de Salto

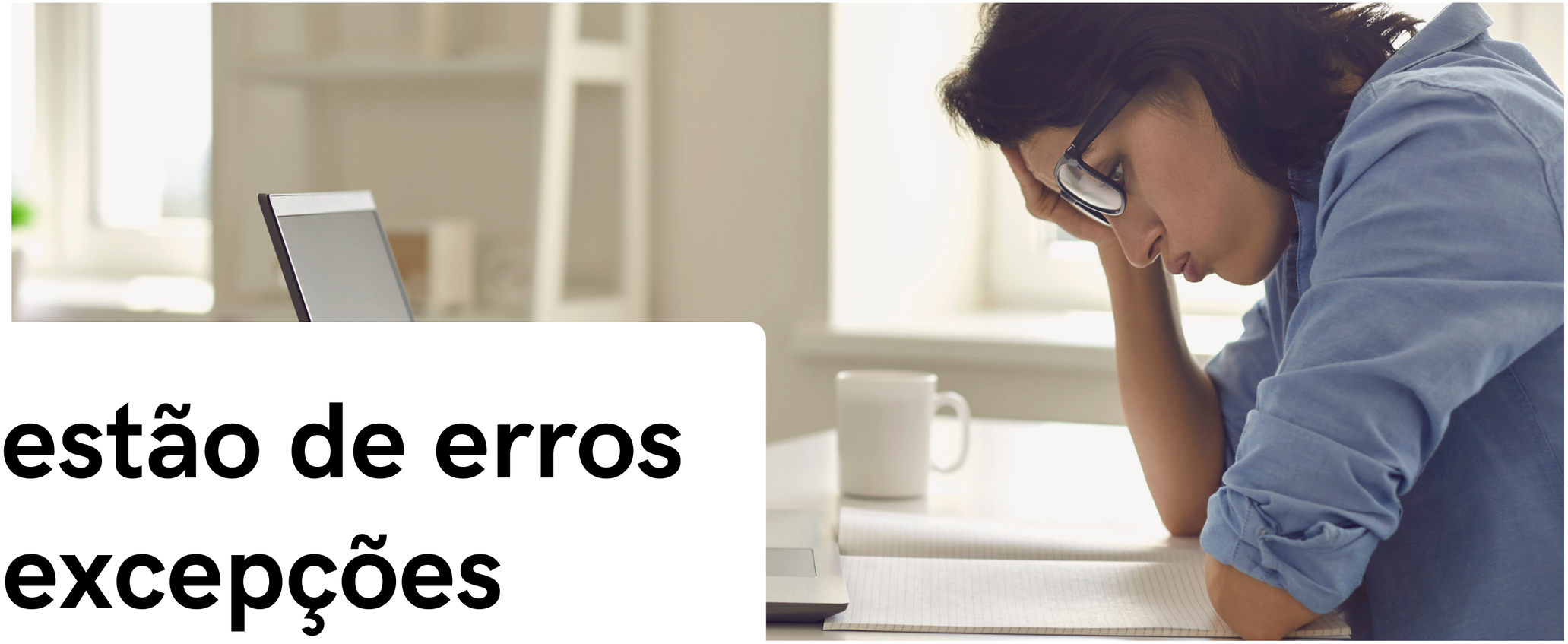
Throw

Indica a ocorrência de uma excepção durante a execução do programa.

Permite adicionar excepções em contextos inesperados

```
string arg = args.Length >= 1 ? args[0] :
throw new ArgumentException("You
must supply an argument");
```

cencal centro de formação profissional para a indústria de cerâmica



Gestão de erros e excepções

Gestão de erros e excepções

Devem ser prevenidos no nosso programa para:

- Evitarmos que estes parem ou que tenham resultados inesperados
- Registar eventos de erros
- Permitir que estes continuem uma correta execução após surgirem.

As exceções são "tratadas" através de blocos try-catch-finally que controlam como o programa prossegue quando o erro ocorre.

Gestão de erros e excepções

```
try
{
    // put the code here that may raise exceptions
}
catch
{
    // handle exception here
}
finally
{
    // (OPTIONAL) final cleanup code
}
```

Gestão de erros e excepções

A gestão especifica de erros, é útil quando pretendemos lógica quando o erro acontece (ex.: mostrar uma mensagem de erro).

```
(...)
catch (IndexOutOfRangeException)
{
    Console.WriteLine("Error: Index should be from 0 to 5.");
}
catch (Exception e)
{
    Console.WriteLine("Error: You did not enter an integer.");
}
```