Universidade Federal do Paraná

Engenharia de Produção

Notas de aula

Informática para Engenharia de Produção

Modelo de objetos do Excel

Agnelo Denis Vieira

2018

Breve histórico da programação

Nota: Grande parte do conteúdo desta seção foi extraído, quase na íntegra, de ([Mabesi](#_ENREF_2)). Pequenas alterações foram realizadas no texto original. Os segmentos recuados à direita e à esquerda foram extraídos desta referência.

Desde o surgimento da computação, as técnicas de programação e desenvolvimento de sistemas sofrem modificações, geralmente para melhor, a cada período de tempo.

No início os computadores eram programados diretamente no hardware, através de fios e interruptores, que eram conectados de acordo com a ocasião. Com a evolução dos equipamentos, e o consequente aumento das necessidades de processamento, surgiram as primeiras linguagens de programação, ainda de baixo nível, como o Assembly, de difícil compreensão, por ser muito próximo da linguagem do computador.

figura conexao eletrica e codigo assembly

Para vencer esta dificuldade começaram a ser desenvolvidas as linguagens de alto nível, mais próximas da linguagem humana, cuja sintaxe de comandos são palavras, inteiras ou abreviadas, em inglês. Até aí os programas eram desenvolvidos de forma linear, um comando depois do outro em um único bloco de código, e assim eram executados. Era a **programação linear**.

Com o aumento da complexidade dos softwares, pesquisadores começaram a se preocupar com a estrutura dos programas, de forma que ficasse mais fácil criar programas ainda mais complexos, pois a programação linear era muito difícil de ser corrigida, já que o programador frequentemente se perdia em meio aos desvios e loops existentes. A partir do aperfeiçoamento e adequação das linguagens de alto nível, tornou-se possível desenvolver programas baseados em funções agrupadas em módulos, onde cada módulo era responsável por determinadas funções, e o conjunto de módulos formava o sistema completo. Eis que surge a **programação estruturada** (PE). Este tipo de abordagem seguiu dois caminhos principais, a programação orientada a funções, onde a base do projeto do software eram os processos necessários à execução das tarefas, e a programação orientada a dados, no qual o projeto do sistema era baseado no modelo de dados. Alguns autores acreditam que a abordagem nos dados é melhor, pois funções são baseadas nas regras do negócio, e regras mudam constantemente, já os dados sofrem muito menos alterações no mesmo período de tempo, por isso apresentam maior consistência.

O paradigma estruturado resolveu grande parte dos problemas por um bom período. As críticas apareceram quando desenvolvedores do mundo todo perceberam que ainda era muito difícil realizar manutenção ou extensão dos softwares estruturados. Não se chega a um consenso sobre o motivo. Alguns afirmam que isto se deve à falta de documentação, outros alegam falta de preocupação com a análise e o projeto. Contudo, todos concordam que ele não conseguia atender à crescente demanda por sistemas de grande complexidade, e que exigiam prazos reduzidos para conclusão, além de possibilitar a manutenção e a extensão de pontos específicos do sistema sem que isto afetasse tanto os outros componentes. Em grande parte dos sistemas estruturados, seria mais fácil recomeçar do zero do que realizar uma alteração bem sucedida.

A proposta que alcançou o nível mais próximo do ideal foi a de tratar os sistemas como eles realmente são. Qualquer software é, na verdade, uma representação da realidade, e o mundo real é composto de objetos independentes que se relacionam. Os **objetos** possuem características (**propriedades**) e realizam ações (**métodos ou funções**) próprias, que os diferenciam dos demais. Alguns objetos possuem características e realizam ações similares, por isso podem ser facilmente agrupados em uma **classe de objetos**, um modelo que serve de referência para qualquer objeto deste tipo.

A partir do estudo dos objetos surge a **programação orientada a objetos** (POO), um novo paradigma cuja ideia principal é a de modelar os sistemas considerando que cada classe de objetos possui características (dados) e realizam ações (funções) próprias.

O sucesso da POO se deve ao fato de que representando o problema através de objetos fica bem mais fácil analisar e compreender a complexidade de qualquer sistema, tanto os mais simples quanto os mais complexos, os quais a PE não conseguia representar com a fidelidade e a simplicidade desejadas. Na POO uma alteração realizada em uma classe geralmente não afeta o restante do software, então a manutenção ou a extensão se resume a alterar o ponto desejado.

Uma das principais vantagens da programação orientada a objetos é, com certeza, a possibilidade de reutilização de códigos em vários sistemas distintos. Assim como já é de praxe em outras áreas do conhecimento, a padronização permite que componentes genéricos de software sejam construídos e possam ser utilizados diretamente em uma infinidade de outros sistemas. Algumas vezes a utilização direta não é possível, mas nestes casos em geral basta uma pequena adaptação.

Vamos ilustrar agora um pouco dos conceitos essenciais da OO. Uma bicileta pode ser descrita por diversas características (ou na terminologia da POO, propriedades): número de série, tamanho do quadro, diâmetro do aro, cor, tipo, entre outras. Por exemplo ela pode ser uma 26" azul com 10 marchas. Uma bicicleta pode ser capaz de realizar diversas ações (na terminologia da POO, métodos), ela pode andar em linha reta ou fazer curvas, ela também pode freiar. Uma bicicleta como um objeto, é por sua vez constituída por outros objetos, como por o exemplo, o quadro, as rodas, o selim. Uma roda por sua vez pode ser constituída por outros objetos, aro, raios, pneu, câmera de ar, eixo, rolamentos. Isto define uma hierarquia construtiva, dos objetos de mais baixo nível (eixo) aos objetos de mais alto nível hierárquico (a bicicleta). Nesta hierarquia cada objeto possui **propriedades** e **métodos** específicos. Na terminologia da POO há ainda os **eventos**, que correspondem a fenômenos instantâneos que disparam a execução de métodos específicos.

Na OO são definidas as **classes de objetos**, que estabelecem de forma geral as propriedades, métodos e eventos. No exemplo acima, bicicleta é uma classe de objetos. Quando são definidos valores específicos para cada uma das suas propriedades é realizada a **instância de um objeto**. A bicicleta "número de série 001, 26" azul com 10 marchas" é uma instância ou um objeto da classe bicicleta.

Todas as possíveis bicicletas possuem os mesmos métodos, propriedades e eventos. Diz-se que, quando um objeto é instanciado (criado), ele herda todos estes métodos, propriedades e eventos da respectiva classe.

Uma cadeira de rodas é muito diferente de uma bicicleta, mas elas podem possuir características em comum. Em particular o objeto roda pode ser utilizado tanto em uma bicicleta quanto em uma cadeira de rodas. Se a classe de objetos roda tiver sido concebida pensando na classe de objetos bicicleta, ela pode ser integralmente reutilizada pensando na classe de objetos cadeira de rodas. Esta característica é denominada reutilização.

Com a criação de padrões internacionais, como no caso da POO, ocorre uma maior facilidade de envolvimento entre desenvolvedores do mundo todo, acelerando tanto o desenvolvimento de sistemas complexos quanto a disseminação de novas técnicas para solução de problemas até então obscuros.

Uma outra vantagem dos sistemas OO é que, em geral, possuem uma divisão de código um pouco mais lógica e melhor encapsulada do que a empregada nos sistemas não orientados a objetos, o que torna a manutenção e extensão do código mais fácil e com menos riscos de inserção de erros. Também é mais fácil reaproveitar o código.

É mais fácil gerenciar o desenvolvimento deste tipo de software quando há uma equipe grande. É possível realizar uma especificação UML (Unified Modeling Language) antes de iniciar o desenvolvimento do software em si, e em seguida dividir o sistema em classes e pacotes, e cada membro da equipe pode ficar responsável por desenvolver uma parte do sistema. Com isto o software ganha uma sobrevida maior que os sistemas estruturados, pois a sua manutenção e até mesmo a sua extensão tornam-se mais fáceis de se implementar.

A UML - Linguagem de Modelagem Unificada (do inglês, UML - Unified Modeling Language) é uma linguagem-padrão para a elaboração da estrutura de projetos de software. Ela poderá ser empregada para a visualização, a especificação, a construção e a documentação de artefatos que façam uso de sistemas complexos de software. Em outras palavras, na área de Engenharia de Software, a UML é uma linguagem de modelagem que permite representar um sistema de forma padronizada (com intuito de facilitar a compreensão pré-implementação). A UML é adequada para a modelagem de sistemas, cuja abrangência poderá incluir desde sistemas de informação corporativos a serem distribuídos a aplicações baseadas na Web e até sistemas complexos embarcados de tempo real. É uma linguagem muito expressiva, abrangendo todas as visões necessárias ao desenvolvimento e implantação desses sistemas. <Wikipédia, 2018 #316>

O maior dos entraves para a adoção da POO nos sistemas atuais está no aprendizado dos conceitos referentes à metodologia, que são complexos e requerem maturidade e massificação de conhecimento do indivíduo.

A correta aplicação dos conceitos requer bastante prática e esforço mental para assimilação e abstração das características do mundo real, o que muitos não estão dispostos a enfrentar. Ao contrário da programação procedural tradicional, na qual basta decorar meia dúzia de comandos e você já consegue fazer um programa simples. Na POO conceitos como herança e polimorfismo, entre outros, geralmente causam um "nó na cabeça" dos iniciantes.

Além do problema do aprendizado, a programação orientada a objetos exige mais do hardware, como capacidade de processamento e memória. É claro que a programação orientada a objetos não pode se comparar, em desempenho, com outros tipos de linguagens procedurais ou lineares. Porém com o avanço da tecnologia de hardware esta perda de desempenho é facilmente compensada pela velocidade e capacidade dos atuais processadores e outros dispositivos de hardware.

Devido a sua complexidade, a metodologia orientada a objetos não é indicada para a produção de pequenos sistemas, pois a quantidade de trabalho inicial necessária não compensa o esforço. Porém, na medida em que o software cresce as vantagens começam a aparecer. Quanto maior fica o sistema, mais a metodologia se paga, em todos os aspectos.

Visual Basic for Applications vs Programação Orientada a Objetos

Conforme ([Birnbaum and Vine, 2007](#_ENREF_1)) Visual Basic for Applications (VBA) não se qualifica como uma linguagem orientada a objetos. O desenvolvimento de código nesta linguagem é fortemente baseado em objetos e nos conceitos associados à POO, mas há tecnicalidades que a desqualificam como uma linguagem verdadeiramente orientada a objetos. VBA pode ser considerada como uma linguagem "baseada em objetos". Apesar da linguagem VBA permitir a criação de novas classes ela não incorpora uma característica essencial denominada herança (*inheritance*).

Neste curso iremos utilizar os objetos do Excel e VBA para o desenvolvimento de programas, mas não realizaremos uma programação orientada a objetos no sentido estrito do termo pois não realizaremos a criação de novas classes. Neste curso desenvolveremos a programação estruturada com a subdivisão de problemas e a resolução de cada subproblema através de subrotinas.

O modelo de objeto do Excel

O conteúdo desta seção está fortemente baseado em ([Walkenbach, 2012](#_ENREF_3)). Recomenda-se a leitura do capítulo 4 da referida obra bem como do capítulo 5 do livro ([Birnbaum and Vine, 2007](#_ENREF_1)).

Capítulo 4: Introdução ao Modelo de Objeto do Excel

Chapter 2: The Excel Object Model

Excel é um objeto e ele contém outros objetos. Esses objetos, por sua vez, contêm ainda mais objetos. Em outras palavras, a programação VBA envolve trabalhar com uma hierarquia de objetos. No alto desta hierarquia está o objeto Application, neste caso, o próprio Excel (a mãe de todos os objetos). O objeto Application contém outros objetos. Por exemplo, um objeto Worksheet (planiha) está inserido em um objeto Workbook (arquivo), que está inserido no objeto Application (Excel). Alguns objetos que podem estar contidos em um objeto Worksheet são:

* Comment (Comentário)
* Hiperlink
* Name (Nome)
* PageSetup (configuração de página)
* PivotTable (Tabela Principal)
* Range (Intervalo de Células)

Cada objeto possui propriedades, métodos e eventos. Para visualizar o "modelo de objetos" definido para o Excel acesse o endereço de internet especificado abaixo. Esta página relaciona os diversos objetos do Excel e apresenta informações detalhadas sobre estes itens, a qual pode ser acessada através do Help Online do VBE.

https://msdn.microsoft.com/pt-br/vba/excel-vba/articles/object-model-excel-vba-reference

Exercício)

Esta página é acessada através do Help Online. No VBE selecione Ajuda - Ajuda do Microsoft Visual Basic for Applications, ou simplesmente Fn + F1. Na página internet aberta selecionar Referência de VBA do Excel.

Insira este endereço nos Favoritos do seu navegador de internet. Você irá consultá-la com muita frequência.

Outra forma de obter informações é, no VBE selecionar Exibir - Pesquisador de Objeto, ou simplesmente Fn+F2. Nesta janela especificar a biblioteca sobre a qual deseja obter ajuda (VBA, Excel, Oficce), digitar o nome do objeto na caixa de busca (ao lado do binóculo), localizar o objeto desejado em Classe e posteriormente a propriedade, método ou evento desejado para então selecionar o ícone de ajuda (ponto de interrogação). A figura abaixo ilustra este procedimento.

|  |
| --- |
|  |

Algumas propriedades de objeto são apenas para leitura, o que significa que você pode ver o valor da propriedade, mas não pode mudá-lo.

Exercício)

Na página mencionada anteriormente, localize o objeto Worksheet, verifique sua descrição bem como a relação de propriedades, métodos e eventos. Explore sua descrição, o método Activate, o evento Activate. Observe que o método e o evento possuem o mesmo nome.

Localize o objeto Range, localize sua descrição, propriedades e métodos. Observe que este objeto não possui eventos.

Em particular explore as propriedades Value, Font, ColumnWidth, DisplayFormat, NumberFormat do objeto Range.

Observe que "Font" é uma propriedade do objeto Range. Font também é um objeto e, portanto, também possui propriedades. Identifique quais são as propriedades do objeto Font.

Explore o método ClearContents do objeto Range.

Propriedades de objetos

Todo objeto tem propriedades. Você pode pensar em propriedades como atributos que descrevem o objeto. A propriedade de um objeto determina sua aparência, como ele age e até se ele é visível. Usando o VBA, você pode fazer duas coisas com as propriedades de um objeto:

* Examinar a configuração atual de uma propriedade.
* Mudar a configuração da propriedade.

Por exemplo, um objeto Range de uma única célula tem uma propriedade chamada Value. A propriedade Value armazena o valor contido na célula.

Você pode escrever o código VBA para exibir a propriedade Value, ou pode escrever o código VBA para atribuir um valor específico à propriedade Value.

Exercício)

Crie um arquivo (Workbook) com uma única planilha (Worksheet) denominada Plan1.

Salve este arquivo com o nome ModObjetos\_SeuNome\_Data.xlsm. Não esqueça de selecionar o tipo de arquivo apropriado.

Especifique um valor qualquer na célula A1.

No VBE, selecione o arquivo recém aberto, insira um novo módulo (Módulo1), no qual você irá inserir as subrotinas dos próximos exercícios.

Implemente no Módulo1 deste arquivo a subrotina que, ao ser executada, apresenta ao usuário uma caixa de texto contendo o valor na célula A1 da Plan1 da pasta de trabalho ativa.

Sub MostraValorCelulaA1()

Dim ValorA1 As Variant

ValorA1 = Worksheets(“Plan1”).Range(“A1”).Value

MsgBox (ValorA1)

End Sub

DICA: Faça uso dos recursos do VBE

Para inserir a linha " ValorA1 = Worksheets(“Plan1”).Range(“A1”).Value"

Digite val - CTRL+Espaço, selecione ValorA1, selecione a tecla TAB

|  |
| --- |
|  |

Digite wor - CTRL+Espaço, selecione Worksheets

|  |
| --- |
|  |

Digite, utilizando apenas letras minúsculas e sem espaço

msgbox(valora1)

Observe que as letras maiúsculas são corrigidas automaticamente e que um espaço é introduzido logo após msgbox

Observe que a variável ValorA1 foi declarada como sendo do tipo Variant para que a célula A1 possa receber qualquer tipo de dado.

Neste mesmo módulo, implemente uma segunda subrotina que recebe um valor através de uma caixa de entrada e estabelece tal valor à célula A1.

Sub EstabeleceValorCelulaA1()

Dim ValorA1 As Variant

ValorA1 = InputBox("Qual valor você deseja atribuir para a célula A1?")

Worksheets(“Plan1”).Range(“A1”).Value = ValorA1

End Sub

ATENÇÃO: Se a pasta de trabalho ativa não tiver uma planilha nomeada Plan1, ao executar a macro será exibida uma mensagem de erro. O VBA apenas segue instruções e não pode trabalhar com uma planilha que não existe.

Renomeie a planilha, para por exemplo Plan2, e execute as subrotinas. Verfique se é apresentada alguma mensagem de erro.

Para renomear a planilha vá no Excel, selecione com o botão direito do mouse a aba com o nome da planilha desejada e selecione Renomear.

|  |
| --- |
|  |

Observe que o erro só é sinalizado quando a linha que está sendo executada é a que faz referência à planilha "Plan1".

Nas subrotinas acima a variável ValorA1 foi declarada como Variant, isto permite que ela receba qualquer tipo de dado. Teste as subrotinas com os seguintes valores:

- 15

- 15.4

- 15.6

- quinze

- Falso

Altere a declaração desta variável para o tipo Integer e teste o programa com os mesmos valores.

Observe que Value é uma propriedade de um objeto Range.

A primeira subrotina recebe o valor desta propriedade e estabelece para a variável ValorA1

ValorA1 = Worksheets(“Plan1”).Range(“A1”).Value

a segunda subrotina recebe o valor de uma variável, ValorA1, e estabelece este valor a esta propriedade

Worksheets(“Plan1”).Range(“A1”).Value = ValorA1

Observe que neste programa você está fazendo referência à objetos.

Plan1 é um objeto, é uma instância da classe Worksheet

A1 é um objeto, é uma instância da classe Range

A hierarquia de objetos é

Application (o Excel)

Workbook (o arquivo que está aberto)

Worksheet (a planilha Plan1)

Range (a célula A1)

Quando você faz referência a um determinado objeto, isto deve iniciar do objeto de nível hierárquico mais elevado (Excel) para o objeto de nível hierárquico mais baixo (A1).

Neste exercício, como há um único arquivo aberto, fica claro durante a execução da subrotina que o arquivo em questão é o arquivo aberto. Então é possível omitir a referência ao arquivo. Como há uma única planilha, também poderia ser omitida a referência à planilha. Exclua do código a referência à planilha, ou seja, digite simplesmente:

ValorA1 =Range(“A1”).Value

a segunda subrotina recebe o valor de uma variável, ValorA1, e estabelece este valor a esta propriedade

Range(“A1”).Value = ValorA1

Ao realizar a referência a um objeto não é permitido "inverter" a sequência hierárquica. O código abaixo irá resultar em erro durante a execução da subrotina

ValorA1 = Range(“A1”).Worksheets(“Plan1”). Value

Worksheets(“Plan1”).Range(“A1”).Value = ValorA1

Você também realizou referência a propriedades de objetos. Value é uma propriedade do objeto Range.

Veja que, você está realizando a programação utilizando conceitos de orientação a objetos. Mas isto não é motivo de alarde.

A linha

ValorA1 = Worksheets("Plan1").Range("A1").Value

é apenas uma forma de informar para o VBE que

"a variável ValorA1 recebe o valor da célula A1 da planilha Plan1"

A linha

Worksheets("Plan1").Range("A1").Value = ValorA1

é apenas uma forma de informar para o VBE

"a célula A1 da planilha Plan1 recebe o valor da variável ValorA1"

Usando a língua portuguesa você poderia escrever estas duas frases de inúmeras formas equivalentes, e o leitor seria capaz de compreender cada uma delas. O compilador de uma linguagem de programação é mais limitado nesta capacidade de compreensão, sendo assim, aprender a programar é aprender a escrever da forma como o compilador consegue compreender.

Métodos de Objeto

Além das propriedades, os objetos têm métodos. Um método é uma ação que você executa com um objeto. Um método pode mudar propriedades de um objeto ou fazer com que o objeto faça alguma coisa.

Alguns métodos tomam um ou mais argumentos. Um argumento é um valor que especifica a ação a ser executada. Você coloca os argumentos para um método depois de fazer referência ao mesmo, separado por um espaço. Argumentos múltiplos são separados por vírgula. Você especifica o nome do argumento e utiliza o comando de atribuição de valor ":=" e o respectivo valor. No exemplo abaixo lê-se "Argumento1 é definido como Valor1".

**Objeto.Método Argumento1 := Valor1, Argumento2 := Valor2**

Se o método possui inúmeros argumentos, é possível omitir o nome e o comando de atribuição de valor, desde que os valores estejam relacionados na sequência de definição do método.

Exercício)

Insira neste arquivo mais uma planilha. Certifique-se que o nome das planilhas sejam Plan1 e Plan2.

No Excel, selecione EXIBIÇÃO - NOVA JANELA

|  |
| --- |
|  |

Observe que você terá o mesmo arquivo aberto em duas janelas distintas. Selecione EXIBIÇÃO - ORGANIZAR TUDO - NA VERTICAL

|  |
| --- |
|  |

Desta forma você irá visualizar simultaneamente as duas planilhas, lado a lado.

|  |
| --- |
|  |

Especifique um valor na célula A1 da planilha Plan1 e DEPOIS um valor na célula A1 da planilha Plan2.

Implemente e execute a subrotina abaixo no Módulo 1 do arquivo que está aberto.

Sub LimpaA1()

Range(“A1”).ClearContents

End Sub

Verifique o conteúdo das células A1 nas duas planilhas.

Especifique um valor na célula A1 da planilha Plan2 e DEPOIS um valor na célula A1 da planilha Plan1.

Execute novamente a subrotina.

Identifique a razão para o conteúdo da célula A1 da planilha Plan2 ter sido eliminado na primeira execução da subrotina enquanto não houve qualquer alteração no conteúdo da célula A1 da planilha Plan1. Já na segunda execução da subrotina o valor eliminado foi o da planilha Plan1 e não o da planilha Plan2.

Insira um novos valores nas célula A1 das planilhas Plan1 e Plan2, nesta sequência.

No Módulo1 do arquivo aberto, implemente e execute a rotina abaixo.

.

Sub CopiaA1paraB1()

Worksheets(“Plan1”).Activate

Range(“A1”).Copy Range(“B1”)

End Sub

No Módulo1 do arquivo aberto, implemente e execute a rotina abaixo.

Sub CopiaA1Plan1paraB1Plan2()

Worksheets("Plan1").Activate

Range("A1").Copy Worksheets("Plan2").Range("B1")

End Sub

Tanto ClearContents quanto Copy são métodos do objeto Range. O primeiro não possui qualquer argumento, o segundo possui um argumento de entrada (Destination) que neste caso é ocional.

Analise a sintaxe de utilização destes dois métodos em

https://msdn.microsoft.com/pt-br/vba/excel-vba/articles/range-object-excel

Sintaxe de utilização

expressão. ClearContents

expressão Uma variável que representa um objeto Range.

expressão . Copy( Destination )

expressão Uma variável que representa um objeto Range.

Parâmetros

Destination (Opcional) Variant Especifica o novo intervalo para o qual o intervalo especificado será copiado. Se esse argumento for omitido, o Microsoft Excel copiará o intervalo para a Área de Transferência.

Estes métodos atuam sobre a planilha ativa. No início da execução do exercício, a ativação da planilha ocorreu quando você selecionou uma planilha para digitar um valor em uma determinada célula. Na segunda subrotina a ativação da planilha ocorre durante a execução da mesma através do método Activate.

Activate é um método do objeto Worksheet.

Sintaxe

expressão . Activate

expressão Uma variável que representa um objeto Worksheet.

Comentários

Chamar este método é equivalente a clicar na guia da planilha.

Eventos de objeto

Evento é algo que acontece instantaneamente e que pode determinar a execução de métodos ou subrotinas.

Exercício)

No arquivo utilizado no exercício anterior, digite valores numéricos quaisquer nas células A1 até A15 da planilha Plan1 (por exemplo 11 até 25) e depois faça o mesmo com outros valores na planilha Plan2 (por exemplo 21 até 35).

Observe que neste momento a planilha ativa é a planilha Plan2.

Até este ponto do exercício, você implementou todas as subrotinas no Módulo1.

|  |
| --- |
|  |

Na sequência você vai implementar subrotinas nas planilhas.

Na janela Project Explorer do VBE selecione a planilha Plan1 (duplo click do mouse). Isto abre uma seção de código específica para esta planilha. A janela de código aparecerá vazia. Nesta janela selecione Worksheet (campo à esquerda) e Activate (campo à direita).

Observe que a estrutura básica de uma subrotina denominada Worksheet\_Activate é automaticamente criada na área para digitação do código. Veja na janela abaixo os elementos especificados.

ATENÇÃO: Você tem liberdade para escolher o nome das subrotinas implementadas em Módulo1, mas você não pode alterar o nome das subrotinas que são automaticamente criadas em uma planilha.

|  |
| --- |
|  |

Complete a definição da subrotina de forma a obter o seguinte resultado

Private Sub Worksheet\_Activate()

Worksheets("Plan1").Range("A1:A10").Sort Key1:=Range("A1"), Order1:=xlAscending

End Sub

Esta subrotina é ativada pela ativação da planilha Plan1, ou seja, toda vez que a planilha é selecionada quando uma outra planilha estava ativa. Quando isto ocorre o Excel sinaliza a ocorrência do evento Activate associado ao objeto Plan1. A ocorrência deste evento determina a execução da subrotina Worksheet\_Activate.

O conteúdo desta subrotina ordena os valores de forma ascendente (do menor para o maior) das células A1 até A10 da planilha Plan1. Observe que os valores das células A11 até A15 não foram reordenados pois foi especificado Range("A1:A10").

Altere livremente valores das células A1 até A15. Observe que os valores ainda não são automaticamente ordenados. Isto ocorre porque a planilha Plan1 "já está ativa". Mas se você selecionar a planilha Plan2, esta passa a ser a planilha ativa, e depois selecionar novamente a planilha Plan1 os valores serão automaticamente. Isto ocorre porque é sinalizada a ocorrência da ativação da planilha Plan1.

Neste ponto, além de realizar uma programação Orientada a Objetos, você também implementou uma programação Orientada a Eventos. A ocorrência do evento "ativação da planilha Plan1" determinou a execução da subrotina Worksheet\_Activate.

O objeto Range pode fazer menção à uma única célula de uma planilha ("A1") ou a um intervalo contendo inúmeras linhas e colunas ("A1:A10").

Sort é um método do objeto Range. Este método possui 15 argumentos, todos opcionais. Neste exercício formam empregados apenas o argumentos Key1 e Order1

Sintaxe

expressão.Sort (Key1, Order1, Key2, Type, Order2, Key3, Order3, Header, OrderCustom, MatchCase, Orientation, SortMethod, DataOption1, DataOption2, DataOption3)

expressão Uma variável que representa um objeto Range.

Key1 (Opcional) Variant Especifica o primeiro campo da classificação, como um nome de intervalo (String) ou um objeto Range; determina os valores como classificados.

Order1 (Opcional) XlSortOrder Determina a ordem de classificação para os valores especificados em Key1

Utilize o Pesquisador de Objeto para descobrir os possíveis valores de XlSortOrder

|  |
| --- |
|  |

Selecione o campo à direta da janela de código para verificar os possíveis eventos associados à uma planilha (Worksheet).

Na verdade, você não precisa usar aquelas duas listas drop-down, mas facilita o seu trabalho, pois o nome do procedimento que lida com evento é muito importante. Se você não tiver o nome exato, ele não funciona. Além disso, alguns procedimentos que lidam com evento usam um ou mais argumentos na declaração Sub. Não há como você lembrar quais são aqueles argumentos.

Coleções

As coleções são um outro tipo de conceito-chave em programação VBA. Uma coleção é um grupo de objetos do mesmo tipo. Aqui estão alguns exemplos das coleções comumente usadas:

Workbooks (Pastas de Trabalho ou Arquivos): Uma coleção de todos os objetos Workbook abertos no momento.

Sheets (Planilhas): Uma coleção de todos os objetos Planilha e Gráficos em um objeto Workbook em especial.

Charts (Gráficos): Uma coleção de todos os objetos Chart (gráficos) contidos em um objeto Workbook em especial.

Worksheets (Planilhas): Uma coleção de todas as planilhas contidas em um objeto Workbook em especial.

Range (intervalo): Uma coleção de intervalos

Você deve perceber que os nomes de coleção estão todos no plural, com exceção de Range.

Coleções são muito úteis quando você deseja fazer algo não apenas em um determinado objeto, com uma planilha, mas com vários objetos da mesma classe. Como você verá, você pode implementar um código em VBA que realiza um conjunto de ações em cada um dos membros de uma coleção.

Uma coleção também é um objeto. Isto significa que uma coleção também tem propriedades e métodos. Por exemplo, você pode determinar como várias pastas de trabalho são abertas acessando a propriedade Count da coleção Workbooks.

Exercício)

No Módulo1 do arquivo em uso implemente a subrotina abaixo. Quando executada ela apresenta o número atual de planilhas existes e cria uma nova planilha com o nome especificado durante a execução

Sub CriaPlanilha()

Dim NumPlan As Integer

Dim NomePlan As String

NumPlan = Worksheets.Count

MsgBox ("O número atual de planilhas é " & NumPlan & ". Será criada uma nova planilha")

NomePlan = InputBox("Especifique o nome da planilha a ser criada")

Worksheets.Add After:=Worksheets(Worksheets.Count)

ActiveSheet.Name = NomePlan

NumPlan = Worksheets.Count

MsgBox "O novo número de planilhas é " & NumPlan

End Sub

Neste exercício Count é uma propriedade do objeto Worksheets, que especifica o número atual de planilhas no arquivo (Workbook) ativo. Add é um método do objeto Worksheets, adiciona uma nova planilha ao arquivo ativo e, portanto, à coleção de planilhas.

Como Fazer Referência aos Objetos

Você pode trabalhar com uma coleção inteira de objetos de uma só vez. No entanto, com mais frequência, é preciso trabalhar com um objeto específico em uma coleção (como uma planilha em especial em uma pasta de trabalho). Há duas opções para fazer referência a um único objeto de uma coleção, você coloca o nome ou o número do índice do objeto entre parênteses depois do nome da coleção. O nome do objeto deve ser colocado entre aspas duplas, enquanto que o número do índice deve ser colocado sem aspas. Se o nome do objeto for colocado sem aspas o Excel interpretará como sendo o nome de uma variável. O índice do objeto ativo é igual a 1, o índice dos demais objetos é variável, sendo alterado conforme são adicionados novos objetos ou quando a disposição física é alterada. Desta forma, quando se deseja especificar um objeto em particular, a referência pelo nome é a forma mais adequada por ser mais precisa. A referência por índice é apropriada quando se deseja acessar todos os objetos de uma determinada coleção através de uma estrutura de repetição. A realização do exercício abaixo permite compreender o conceito de índice de objeto, e suas limitações.

Exercício)

Implemente a subrotina abaixo e execute-a inúmeras vezes alterando a disposição das planilhas no canto inferior esquerdo do Excel.

A cada execução anote o nome, o índice e a respectiva sequência de apresentação na aba de planilhas.

Observe que há alteração da correspondência nome e índice.

Sub TestaIndice()

Dim i As Integer

Dim NumPlan As Integer

Dim IndicePlan As Integer

Dim NomePlan As String

NumPlan = Worksheets.Count

For i = 1 To NumPlan

NomePlan = Worksheets(i).Name

MsgBox ("O nome da planilha de índice " & i & " é " & NomePlan)

Next i

A referência a um objeto pode ser "totalmente qualificada" ou "simplificada. A referência à célula A1 da planilha Plan1 do arquivo Arquivo1 do Excel na forma totalmente qualificada fica como abaixo.

Application.Workbooks("Arquivo1.xlsx").Worksheets("Plan1").Range("A1").Value = 325

Observe que a hierarquia entre os objetos deve ser respeitada, do nível mais elevado para o mais baixo. Além disto, a referência é sempre à coleção correspondente, por isto os nomes estão sempre no plural (Workbooks, Worksheets) com exceção da coleção Range, que não possui plural.

A referência totalmente qualificada tem a vantagem de ser explicita sobre qual objeto está sendo realizada referência. Particularmente se há diversos arquivos abertos e se em cada arquivo há diversas planilhas. Contudo, esta forma de especificação requer a digitação de muito código, particularmente se forem realizadas inúmeras referências ao mesmo objeto em sequência. Neste caso, pode-se tornar os objetos de maior nível hierárquico ativos para então realizar a referência simplificada ao objeto desejado.

Application.Workbooks("ModObjetos\_Agnelo\_29jan2018.xlsm").Worksheets("Plan1").Activate

Range("A2").Value = 521

Lembre que objetos possuem métodos e propriedades. Para realizar referência à propriedades e/ou métodos de um determinado objeto, utiliza-se a notação:

objeto.propriedade = valor (atribuição de valor à propriedade do objeto)

variável = objeto.propriedade (atribuição do valor da propriedade do objeto à variável)

objeto.método(parâmetro1 := valor1, parâmetro2 := valor2, ...) (execução do método do objeto)

Quando é necessário realizar referência à inúmeras propriedades e/ou métodos de um determinado objeto, pode-se utilizar a notação "With ... End With". Neste caso, após especificar o objeto uma única vez, pode-se omitir a referência ao mesmo nas inúmeras referências às propriedades e métodos.

Exercício)

Implemente a subrotina no Módulo1 do arquivo em edição.

Sub FormasDeReferencia()

' Referência totalmente qualificada

Application.Workbooks("ModObjetos\_Agnelo\_29jan2018.xlsm").Worksheets("Plan1").Range("A1").Value = 325

Application.Workbooks("ModObjetos\_Agnelo\_29jan2018.xlsm").Worksheets("Plan1").Range("A1").Font.Name = "Arial"

Application.Workbooks("ModObjetos\_Agnelo\_29jan2018.xlsm").Worksheets("Plan1").Range("A1").Font.Bold = True

Application.Workbooks("ModObjetos\_Agnelo\_29jan2018.xlsm").Worksheets("Plan1").Range("A1").Font.Color = RGB(0, 255, 0)

' Referência simplificada com ativação prévia dos objetos de nível hierárquico superior

Application.Workbooks("ModObjetos\_Agnelo\_29jan2018.xlsm").Worksheets("Plan1").Activate

Range("A2").Value = 521

Range("A2").Font.Name = "Comic Sams MS"

Range("A2").Font.Bold = True

Range("A2").Font.Color = RGB(255, 0, 0)

' Referência a propriedades e métodos de um mesmo objeto com a estrutura With End With na forma totalmente qualificada

With Application.Workbooks("ModObjetos\_Agnelo\_29jan2018.xlsm").Worksheets("Plan1").Range("A3")

.ClearContents

.Value = 752

.Font.Name = "Courier New"

.Font.Bold = False

.Font.Color = vbBlue

End With

' Referencia a propriedades e métodos de um mesmo objeto com a estrutura With End With na forma simplificada

Application.Workbooks("ModObjetos\_Agnelo\_29jan2018.xlsm").Worksheets("Plan1").Activate

With Range("A4")

.Value = 525

With .Font

.Name = "Rockwell"

.Color = vbGreen

.Size = 18

End With

End With

End Sub

Observe o aninhamento das estruturas With ... End With

A mais externa está associada ao objeto Range e a mais interna ao objeto Font.

Neste exemplo a cor da fonte foi especificada, tanto na forma RGB(255, 0, 0) quanto empregando as constantes do VBA vbBlue

Sub PlanilhaAtiva()

Dim NumPlan As Integer

Dim NomePlan As String

NomePlan = ActiveSheet.Name

NumPlan = ActiveSheet.Index

MsgBox ("O nome da planilha ativa é " & NomePlan & " e o seu índice é " & NumPlan)

End Sub

Toda planilha do Excel é um objeto, e desta forma possui propriedades. Neste momento, duas propriedades de nosso interesse são Name e CodeName. A primeira é o nome da planilha que está visível com o Excel aberto. A segunda só é acessível através da janela de Propriedades do VBE. Esta propriedade só pode ser alterada via janela de Propriedades mas o valor desta propriedade pode ser determinado via código implementado em VBA. Isto faz com que a referência à uma planilha pelo CodeName seja um procedimento mais seguro do que pelo Name.

Na figura abaixo, na janela Explorador do Projeto, a planilha "Plan20 (p3)" está selecionada. Na janela Propriedades, o valor da propriedade (Name) é Plan20 e da propriedade Name é p3.

|  |
| --- |
|  |

Exercício

Implemente a subrotina no Módulo 1

Sub NomePlanilhaAtiva()

Dim NumPlan As Integer

Dim NomePlan As String

Dim NomeCodigoPlan As String

NomePlan = ActiveSheet.Name

NomeCodigoPlan = ActiveSheet.CodeName

NumPlan = ActiveSheet.Index

MsgBox ("O Name da planilha ativa é " & NomePlan & " o CodeName é " & NomeCodigoPlan & " e o seu índice é " & NumPlan)

NomePlan = InputBox("Defina o novo Name da planilha ativa")

NomeCodigoPlan = InputBox("Defina o novo CodeName da planilha ativa")

ActiveSheet.Name = NomePlan

' ActiveSheet.CodeName = NomeCodigoPlan Não é possível alterar CodeName via código apenas via janela de Propriedades

End Sub

Referências

Birnbaum, D. and M. A. Vine (2007). Microsoft Excel VBA programming for the absolute beginner, Cengage Learning.

Mabesi, P. "Usando Access ", 2018, from http://www.usandoaccess.com.br/tutoriais/classe-no-access-orientacao-a-objetos.asp?id=1#inicio.

Walkenbach, J. (2012). Programando Excel VBA para leigos, Alta Books.