Introdução ao Go

Aula 36: Entendendo as Declarações "switch"

Nesta aula, vamos aprender como substituir longos blocos de `if-else if` por uma estrutura mais limpa e legível chamada `switch`, que é uma maneira eficiente de lidar com várias condições e executar blocos de código diferentes.

Estrutura do `switch`

A estrutura básica de um `switch` é a seguinte:

```go
switch expression {
    case value1:
        // Código executado quando expression == value1
    case value2:
        // Código executado quando expression == value2
    case value3:
        // Código executado quando expression == value3
    default:
        // Código executado quando expression não corresponde a nenhum dos valores anteriores
}
```

- `switch`: Indica o início da estrutura de controle.
- `expression`: O valor a ser avaliado.
- `case`: Cada valor que será comparado com a expressão. Se a expressão for igual a um valor de `case`, o bloco de código correspondente será executado.
- `default`: Caso nenhum dos `case` seja verdadeiro, o código dentro de `default` será executado. Isso é equivalente ao `else` em uma cadeia de `if-else`.

Exemplo de Uso do `switch`

Imaginemos que temos um código onde queremos verificar a escolha de um usuário. A forma tradicional seria com um bloco `if-else if`, mas podemos simplificar usando `switch`.

go

```go
package main

import "fmt"

func main() {
    choice := 2  // Suponha que o usuário escolheu a opção 2

    switch choice {
    case 1:
        fmt.Println("Você escolheu a opção 1.")
    case 2:
        fmt.Println("Você escolheu a opção 2.")
    case 3:
        fmt.Println("Você escolheu a opção 3.")
    default:
        fmt.Println("Opção inválida.")
    }
}
```

Explicação

- `switch choice`: Estamos avaliando o valor da variável `choice`.
- `case 1`, `case 2`, `case 3`: Cada `case` verifica se o valor de `choice` é igual ao número especificado. Se for, o código dentro do bloco correspondente será executado.
- `default`: Se nenhum dos `case` for verdadeiro (se o usuário escolher um valor que não seja 1, 2 ou 3), o código no `default` será executado.

Uso do `break` no `switch`

Em Go, **não é necessário usar o** `break` após cada `case` como em outras linguagens. Isso ocorre porque, no Go, apenas **um** `case` **pode ser executado**, e o fluxo já sai do `switch` automaticamente após o primeiro `case` correspondente.

Contudo, o `break` ainda pode ser utilizado dentro de um `switch`, mas seu comportamento será diferente: ele será usado para **sair do** `switch` e não de loops, como ocorre em outras situações.

Quando Usar `switch`

O `switch` é útil quando você tem várias condições baseadas em um único valor e deseja evitar um código repetitivo com múltiplos `if-else if`. Além disso, o `switch` pode ser mais claro e fácil de entender em situações em que há muitas comparações para fazer com um valor.

Se você precisar de um **controle de fluxo mais complexo**, como em loops, o `switch` pode não ser a melhor opção, e você pode continuar com o uso do `if-else` ou outros tipos de controle.

Próximos Passos

- **Explorar o uso de** `switch` **com variáveis mais complexas**, como strings ou expressões.
- **Testar o** `default` para lidar com entradas inesperadas e garantir que seu código tenha um comportamento previsível.

Se quiser mais exemplos ou tiver dúvidas, só avisar!

Para adaptar o código do **Go Bank** utilizando o `switch`, podemos substituir a série de `if-else if` por um `switch`, que torna o código mais claro e conciso. Aqui está uma versão modificada do código usando `switch`:

Código Adaptado com `switch`:

```go
```

```go
package main

import (
    "fmt"
)

func main() {
    var choice int
    var balance float64

    // Mensagem de boas-vindas
    fmt.Println("Bem-vindo ao nosso banco!")
    fmt.Println("Escolha uma opção:")
    fmt.Println("1 - Verificar saldo")
    fmt.Println("2 - Depositar dinheiro")
    fmt.Println("3 - Retirar dinheiro")
    fmt.Println("4 - Sair")

    // Entrada da opção do usuário
    fmt.Scanln(&choice)

    // Estrutura switch para lidar com as escolhas
    switch choice {
    case 1:
        fmt.Printf("Seu saldo é: R$ %.2f\n", balance)
    case 2:
        var deposit float64
        fmt.Print("Digite o valor para depósito: R$ ")
        fmt.Scanln(&deposit)
        balance += deposit
        fmt.Printf("Depósito de R$ %.2f realizado com sucesso. Saldo atual: R$
%.2f\n", deposit, balance)
    case 3:
        var withdraw float64
        fmt.Print("Digite o valor para retirada: R$ ")
        fmt.Scanln(&withdraw)
        if withdraw > balance {
            fmt.Println("Erro: saldo insuficiente!")
        } else {
            balance -= withdraw
            fmt.Printf("Retirada de R$ %.2f realizada com sucesso. Saldo atual: R$
%.2f\n", withdraw, balance)
```

```
        }
    case 4:
        fmt.Println("Obrigado por escolher nosso banco. Até logo!")
        return // Finaliza o programa
    default:
        fmt.Println("Opção inválida! Tente novamente.")
    }

    // A aplicação continuará executando até que o usuário escolha a opção 4 para
sair
}
```

O que mudou?

1. **Substituição de `if-else if` por `switch`**:
   - O bloco `switch choice` é usado para comparar a variável `choice` com os diferentes valores das opções (1, 2, 3, 4). Dependendo do valor de `choice`, o programa executa a operação correspondente.
2. **Adição do `default`**:
   - O `default` cobre casos em que o usuário insere um número fora das opções disponíveis. Isso é equivalente ao `else` no bloco `if-else if`.
3. **Uso do `return`**:
   - Quando a opção 4 é escolhida (para sair), o comando `return` encerra o programa, fazendo com que ele saia da função `main` e termine a execução.

Benefícios do `switch`:

- **Leitura e manutenção do código**: O uso de `switch` facilita a leitura, pois as opções ficam mais organizadas.
- **Menos código redundante**: Não é necessário escrever múltiplos `else if`, tornando o código mais compacto.

Se tiver alguma dúvida sobre como esse código funciona ou quiser fazer ajustes, fique à vontade para perguntar!