

Controle de fluxo com switch

Transcrição

Além do `if`, sabemos que existem outras instruções para controlarmos o fluxo da nossa aplicação. E uma que se adequa melhor ao nosso código é a instrução `switch`.

Essa instrução recebe uma variável e dá possíveis situações para cada valor dessa variável. Nós dizemos quais são os casos, e para cada um há uma situação. Caso a variável valha 0, acontece algo, caso valha 1, acontece outra, e assim por diante.

```
switch comando {
case 1:
    fmt.Println("Monitorando...")
case 2:
    fmt.Println("Exibindo Logs...")
case 0:
    fmt.Println("Saindo do programa...")
}
```

Mas se o valor da variável não estiver em nenhum dos casos listados? Para isso, existe o caso `default`, que é o que será executado se os nossos casos não forem atendidos:

```
package main

import "fmt"

func main() {
    nome := "Douglas"
    versao := 1.1
    fmt.Println("Olá, sr.", nome)
    fmt.Println("Este programa está na versão", versao)

    fmt.Println("1- Iniciar Monitoramento")
    fmt.Println("2- Exibir Logs")
    fmt.Println("0- Sair do Programa")

    var comando int
    fmt.Scan(&comando)
    fmt.Println("O comando escolhido foi", comando)

    switch comando {
    case 1:
        fmt.Println("Monitorando...")
    case 2:
        fmt.Println("Exibindo Logs...")
    case 0:
        fmt.Println("Saindo do programa...")
    default:
        fmt.Println("Não conheço este comando")
    }
}
```


Ao executar o programa, ele funciona como antes, mas desta vez com uma nova instrução de controle de fluxo.

Uso do `break`

Para quem vem de outras linguagens de programação, pode estranhar o não uso do `break`, ao final do código de cada caso do `switch`. O `break` serviria para evitar a execução do código de mais de um caso, se mais de um caso for atendido.

No Go, ele não possui o `break`, pois somente um caso pode ser atendido. O primeiro caso que for atendido, terá o seu código executado e o `switch` será encerrado.