Documentação do Projeto: SWIFT

Alunos:

Natan Batalha de Araújo

Gustavo Horning

Ricardo Hammerschmidt

Lógica de Construção:

1) Criação de Protocolo ***Notificável*** com a propriedade mensagem e o método enviarNotificação().

Modelagem de Struct para diferente tipos de canais: E-mail, SMS e "push notification"

Cada um desses é um tipo de dados diferentes, mas todos devem seguir o mesmo protocolo, a mesma modelagem, o qual seria o notificável, e usarmos Struct para isso.

Ele organiza os dados que cada tipo precisa e permite definir o comportamento de cada tipo.

2) O Segundo passo foi dar continuidade as demais *struct* do canal:

Já temos o Email, continuaremos para SMS e "Push Notification", os quais irão seguir o mesmo protocolo Notificável, cada um com sua propriedade exclusiva.

```swift
//Continuidade dos demais Structs, de SMS e "Push Notification"

struct SMS: Notificavel{
  var mensagem: String
  var numeroTelefone: String

  func enviarNotificação(){
    print("Enviando SMS para \(numeroTelefone): \(mensagem)")
  }
}

struct pushNotification: Notificavel{
  var mensagem: String
  var tokenDispositivo: String

  func enviarNotificação(){
    print("Enviando push para token \(tokenDispositivo): \(mensagem)")
  }
}

```

3) Aperfeiçoamento do sistema de mensagens usando ENUM:

Definimos o enum que deve representar os tipos possíveis de mensagens que podem ser enviadas. Os casos que definimos ajudam a caracterizar notificações de acordo com seu propósito

case promoção – Usado para mensagens de ofertas, descontos, ações de marketing

case lembrete – Notificações com lembretes importantes para o usuário

case alerta – Avisos urgentes ou críticos que exigem atenção imediata

O novo Struct que fizemos na imagem abaixo, representa o conteúdo de uma mensagem enviada por um canal de notificação, combinando o tipo da mensagem com o texto a ser enviado.

```
51
52  //Vamos criar enum com os tipos de mensagens
53
54  enum tipoMensagem{
55    case promocao
56    case lembrete
57    case alerta
58  }
59
60  //Mais um strcut que vai representar a mensagem
61
62  struct Mensagem {
63    var tipo: tipoMensagem
64    var conteudo: String
65  }
66
67  let msg1 = Mensagem(tipo: .promocao, conteudo: "Cupom de 20% OFF pra você!")
68  let msg2 = Mensagem(tipo: .lembrete, conteudo: "Não esqueça sua consulta amanhã.")
69  let msg3 = Mensagem(tipo: .alerta, conteudo: "Tentativa de login suspeita!")
70
71  let email = Email(mensagem: msg1, enderecoEmail: "natan.advpr@gmail.com")
72  let sms = SMS(mensagem: msg2, numeroTelefone: "+5541988653982")
73  let push = PushNotification(mensagem: msg3, tokenDispositivo: "abc123xyz")
74
75  email.enviarNotificacao()
76  sms.enviarNotificacao()
77  push.enviarNotificacao()
78
```

4) Definimos a importância de criar um Array que guarda vários canais de notificação diferentes (Email, SMS, Push), desde que todos conformem ao protocolo principal (“Notificável”) junto alinhando uma lógica de execução com loop

```
76  //Mais um strcut que vai representar a mensagem
77  //Primeiro array com a finalidade de ter canais que irão seguir o protocolo Notificavel
78
79  let canais: [Notificavel] = [email, sms, push]
80
81  for canal in canais{
82    canal.enviarNotificacao()
83  }
```

Como somos alunos dedicados, iremos entregar mais do que é pedido. Definimos os desafios adicionais.

5) Vamos definir as prioridades das notificações.

Nesse caso decidimos criar um novo Enum Prioridade com os casos: baixa, média e alta.

Depois não poderíamos não colocar no protocolo Notificável e ATUALIZAR os canais (Email, SMS e PushNotification) para considerar a prioridade na hora de enviar a notificação.

e como solicitado na atividade, fazer com que as solicitações de **alta prioridade** mostrem **URGENTE** no console.

```swift
//Vamos criar enum com os tipos de mensagens

enum tipoMensagem: String {
  case promocao
  case lembrete
  case alerta
}

struct Mensagem {
  var tipo: tipoMensagem
  var conteudo: String
}

//Enum para solicitações de alta prioridade

enum Prioridade{
  case baixa
  case media
  case alta
}
```

Fizemos as modificações para incluir no protocolo notificável:

```swift

// Definindo o protocolo notificável

protocol Notificavel {
  var mensagem: Mensagem {get set}
  var prioridade: Prioridade {get set}
  func enviarNotificacao()
}

extension Notificavel{
  func enviarNotificacao(){
    print("Enviando notificação...")
  }
}

```

Por fim, iremos definir a lógica para o **Filtro por tipo de canal:**

Iremos criar uma Function que recebe um array de notificável e um tipo específico de canal e retorna os canais do tipo escolhido.

Essa parte mostra que entendemos os conceitos ensinados pelo professor sobre “protocol composition” e “type casting”

6) Por fim fizemos as demais filtragem para os outros tipos de canais e deixamos comentados caso o professor solicite

```
105
106 //Testando o filtro de canal escolhido
107
108 print("\n--- Filtrando apenas canais de Email ---\n")
109
110 let somenteEmails = filtrarCanais(de: Email.self, em: canais)
111
112 for email in somenteEmails {
113     email.enviarNotificacao()
114 }
115
116 //observação, caso o professor solicite outros exemplos de filtragem como SMS e Push Notification segue o raciocónio
    comentado
117
118 // Exemplo: Filtrar apenas canais de SMS
119 // print("\n--- Filtrando apenas canais de SMS ---")
120 // let apenasSMS = filtrarCanais(de: SMS.self, em: canais)
121 // for sms in apenasSMS {
122 //     sms.enviarNotificacao()
123 // }
124
125 // Exemplo: Filtrar apenas canais de PushNotification
126 // print("\n--- Filtrando apenas canais de Push Notification ---")
127 // let apenasPush = filtrarCanais(de: PushNotification.self, em: canais)
128 // for push in apenasPush {
129 //     push.enviarNotificacao()
130 // }
131
132
```