Instituto de Computação

Serviços de atendimento da Polícia Federal

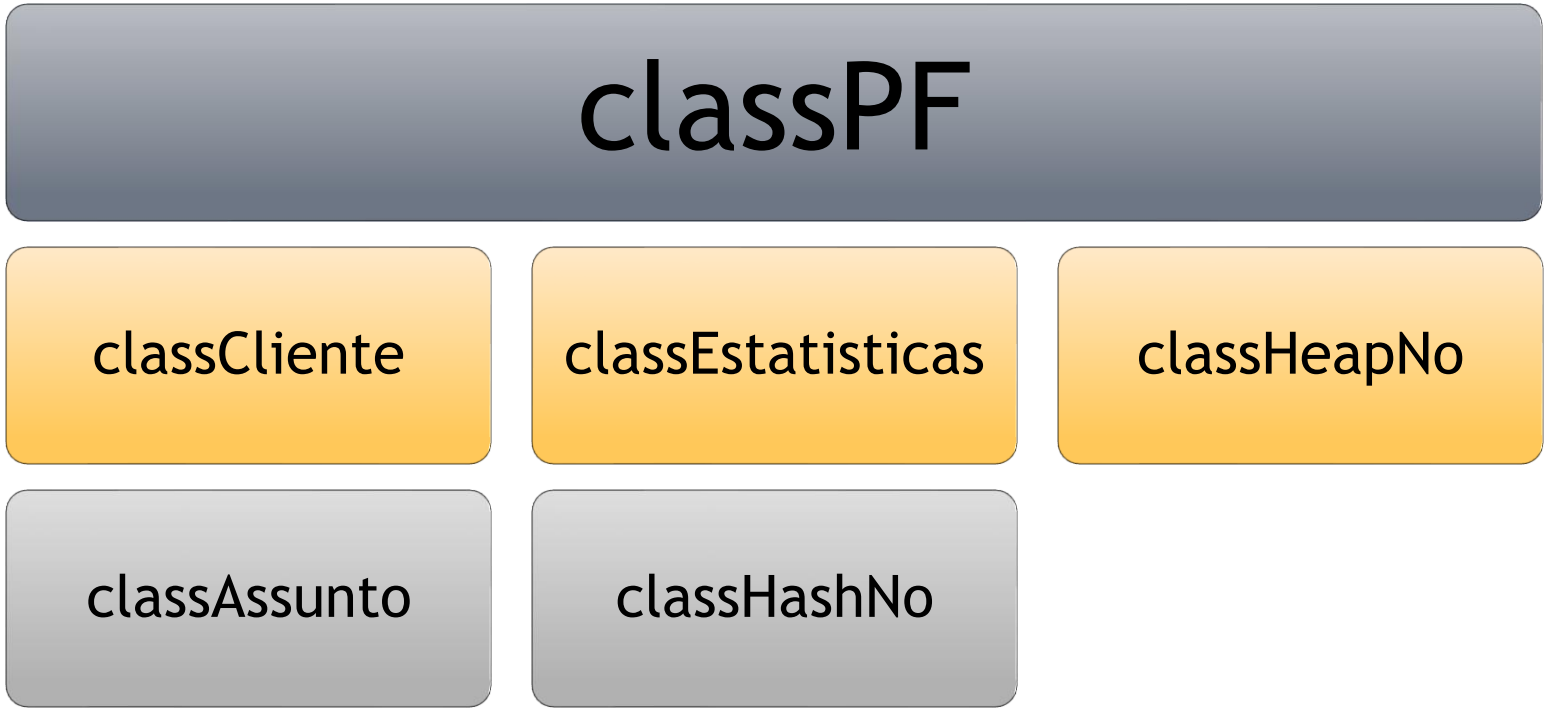
Trabalho de implementação

Disciplina: Estrutura de dados e algoritmos

Aluna: Carolina Veiga Ferreira de Souza

Matrícula: D022.118.004

Classes


classPF
classCliente
classEstatisticas
classHeapNo
classAssunto
classHashNo

classPF

```
from classCliente import Cliente
from classEstatisticas import Estatisticas
from classHeapNo import Heap
from datetime import datetime
from time import sleep

class PF(object):
    def __init__(self):
        self.fila = Heap()
        self.estatisticas = Estatisticas()

    def recepcionar(self, cliente):

    def atender(self):

    def encerrar(self, proximo):

    def gerarEstatisticas(self):
```

def recepcionar(self, cliente):

```
self.cliente = cliente
self.horaChegada = datetime.now()
self.fila.inserir(self.cliente, self.horaChegada)
```

def atender(self):

```
if self.fila.isEmpty() == True:
    print('\n\033[33mAtendimentos encerrados.\033[m')

else:
    proximo = self.fila.remover()
    proximo.horaAtendimento = datetime.now()

    for i in range(0, proximo.cliente.nAssuntos):
        proximo.duracaoAtendimento += proximo.cliente.assuntos[i].tempomin

    for i in range(0, proximo.cliente.nAssuntos):
        self.estatisticas.contarTempo(proximo.cliente.assuntos[i].tipo,
proximo.cliente.assuntos[i].tempomin)
    sleep(proximo.duracaoAtendimento)
    self.encerrar(proximo)
```

def encerrar(self):

```
def encerrar(self, proximo):
    print(f'\n\033[32mProvidencia(s) para {proximo.cliente.nome}:\n')
    for i in range(0, proximo.cliente.nAssuntos):
        print(f'\033[32m{i+1}: {proximo.cliente.assuntos[i].providencia}')
```

def gerarEstatisticas(self):

```
def gerarEstatisticas(self):

    self.estatisticas.construirHash()
    self.estatisticas.imprimirEstatisticas()
```

classHeapNo

```
class No(object):

    def __init__(self, cliente, horaChegada):
        self.cliente = cliente
        self.horaChegada = horaChegada
        self.horaAtendimento = None
        self.duracaoAtendimento = 0
        self.prioridade = self.calcularPrioridade()

    def calcularPrioridade(self):
        self.espera = (datetime.now() - self.horaChegada).total_seconds()
        self.prioridade = ((self.cliente.idade / 65) + (self.espera / 15) +
(self.cliente.urgencia / 10)) / 3
```

classHeapNo

```
class Heap(object):

    def __init__(self):
        self.vetor = []
        self.n = len(self.vetor)

    def isEmpty(self):

    def inserir(self, cliente, horaChegada):

    def descer(self, i, fim):

    def remover(self):

    def reconstruirHeap(self):

    def estadoHeap(self):
```

def inserir(self, cliente, horaChegada):

```
def inserir(self, cliente, horaChegada):
    no = No(cliente, horaChegada)
    self.vetor.append(no)
    self.n = len(self.vetor)
    self.reconstruirHeap()
```

def reconstruirHeap(self):

```
def reconstruirHeap(self):

    for i in range(0, len(self.vetor)):
        self.vetor[i].calcularPrioridade()

    for i in range(int(self.n / 2), -1, -1):
        self.descer(i, self.n)
```

classEstatisticas

```
from classHashNo import Hash, No

class Estatisticas(object):

    def __init__(self):
        self.h = Hash()
        self.tempos = [0]*5
        self.cont = [0]*5
        self.no1 = No('tipo1')
        self.no2 = No('tipo2')
        self.no3 = No('tipo3')
        self.no4 = No('tipo4')
        self.no5 = No('tipo5')

    def contarTempo(self, tipo, tempo):
    def construirHash(self):
```

def contarTempo(self, tipo, tempo):

```
def contarTempo(self, tipo, tempo):

    if tipo == 'tipo1':
        self.tempos[0] += tempo
        self.cont[0] += 1
    elif tipo == 'tipo2':
        self.tempos[1] += tempo
        self.cont[1] += 1
```

def construirHash(self):

```
def construirHash(self):

    for i in range(0, 5):
        media = self.tempos[i]/self.cont[i]
        if i == 0:
            self.no1.mediaTempos = media
            self.h.inserir(self.no1)
```

classHashNo

```
class No(object):
    def __init__(self, tipo):
        self.tipo = tipo
        self.mediaTempos = 0

class Hash(object):

    def __init__(self):
        self.n = 10
        self.hash = [None] * 10
        self.cont = 0
```

```
    def funcaoHash(self, x):

    def isFull(self):

    def isEmpty(self):

    def inserir(self, novo):

    def buscar(self, no):

    def remover(self, no):

    def estadoHash(self):

    def verificarColisoes(self):
```

def funcaoHash(self, x):

```
def funcaoHash(self, x):
    i = round(x.mediaTempos) % 7
    return i
```

def inserir(self, novo):

```
def inserir(self, novo):
    if self.isFull() == True:
        print(f'Hash cheio. {novo} nao incluido')
        pass
    else:
        i = self.funcaoHash(novo)
        if self.hash[i] == None:
            self.hash[i] = novo
        else:
            i = self.n - 1
            while not self.hash[i] == None:
                i -= 1
                if i == -1:
                    i = self.n - 1
            self.hash[i] = novo

        self.cont += 1
```

classCliente – importa classAssuntos

- Construtor:

```
def __init__(self, cpf, nome, idade, vetorAssuntos):
        self.cpf = cpf
        self.nome = nome
        self.idade = idade
        self.assuntos = self.gerarAssuntos(vetorAssuntos)
        self.nAssuntos = len(self.assuntos)
        self.urgencia = self.calculaUrgencia(self.assuntos)
```

- Métodos:

```
def gerarAssuntos(self, vetorAssuntos):
    assuntos = []

    for i in range(0, len(vetorAssuntos)):
        assuntos.append(Assunto(vetorAssuntos[i]))
    return assuntos

def calculaUrgencia(self, assuntos):
    assuntos = assuntos

    urgenciavetor = []
    for i in range(0, len(assuntos)):
        urgenciavetor.append(assuntos[i].urgencia)

    urgenciaMedia = sum(urgenciavetor) / len(urgenciavetor)

    return urgenciaMedia
```

classAssunto

- Tipo1:
  - Informações, sugestões ou críticas
- Tipo2:
  - Cancelar/consultar agendamento
  - Agendar/Reagendar fotografia
  - Emissão de GRU
- Tipo3:
  - Requerimento de passaporte/passaporte eletrônico/estrangeiro/emergência
  - Comunicação de ocorrência com documentos de viagem
- Tipo4:
  - Aquisição/Registro/Renovação de registro/Porte/Tranferência de arma de fogo
  - Comunicação de ocorrência com arma de fogo
- Tipo5:
  - Emissão de certidão de antecedentes criminais
  - Credenciamento de instrutores de armamento e tiro

classAssunto

```
def __init__(self, qualassunto):

    self.titulo = qualassunto

    if self.titulo == 'informacoes' or self.titulo == 'Sugestoes' or self.titulo == 'Criticas':
        self.tipo = 'tipo1'
        self.urgencia = 0
        self.tempomin = randint(1,3)
        if self.titulo == 'informacoes':
            self.providencia = 'Informacoes concedidas.'
        else:
            self.providencia = f'{self.titulo} registradas.'

    elif self.titulo == 'cancelar agendamento' or self.titulo == 'consultar agendamento':
        self.tipo = 'tipo2'
        self.urgencia = 1
        self.tempomin = randint(2,5)
        if self.titulo == 'cancelar agendamento':
            self.providencia = 'Agendamento cancelado com sucesso.'
        else:
            self.providencia = 'Agendamento consultado com sucesso.'
```

classAssunto

```
elif self.titulo == 'agendar fotografia' or self.titulo == 'reagendar fotografia':
    self.tipo = 'tipo2'
    self.urgencia = 2
    self.tempomin = randint(2,5)
    if self.titulo == 'agendar fotografia':
        self.providencia = 'Fotografia agendada com sucesso.'
    else:
        self.providencia = 'Fotografia reagendada com sucesso.'

elif self.titulo == 'emissao de GRU':
    self.tipo = 'tipo2'
    self.urgencia = 3
    self.tempomin = randint(2, 5)
    self.opcoesprovidencias = ['GRU emitido com sucesso.', 'Problemas de autenticacao.']
    self.providencia = self.opcoesprovidencias[randint(0,1)]
```