Programação Orientada Objeto

https://dontpad.com/catolicaoop2025

https://meet.google.com/ccx-fgtf

Dia 19 de Agosto de 2025 a aula começou com o professor relembrando o que fora passado na aula passada, foi dado um exemplo de abstração de um elevador na qual abstraímos informações como andar máximo, andar mínimo, andar atual, se a porta está aberta ou não.

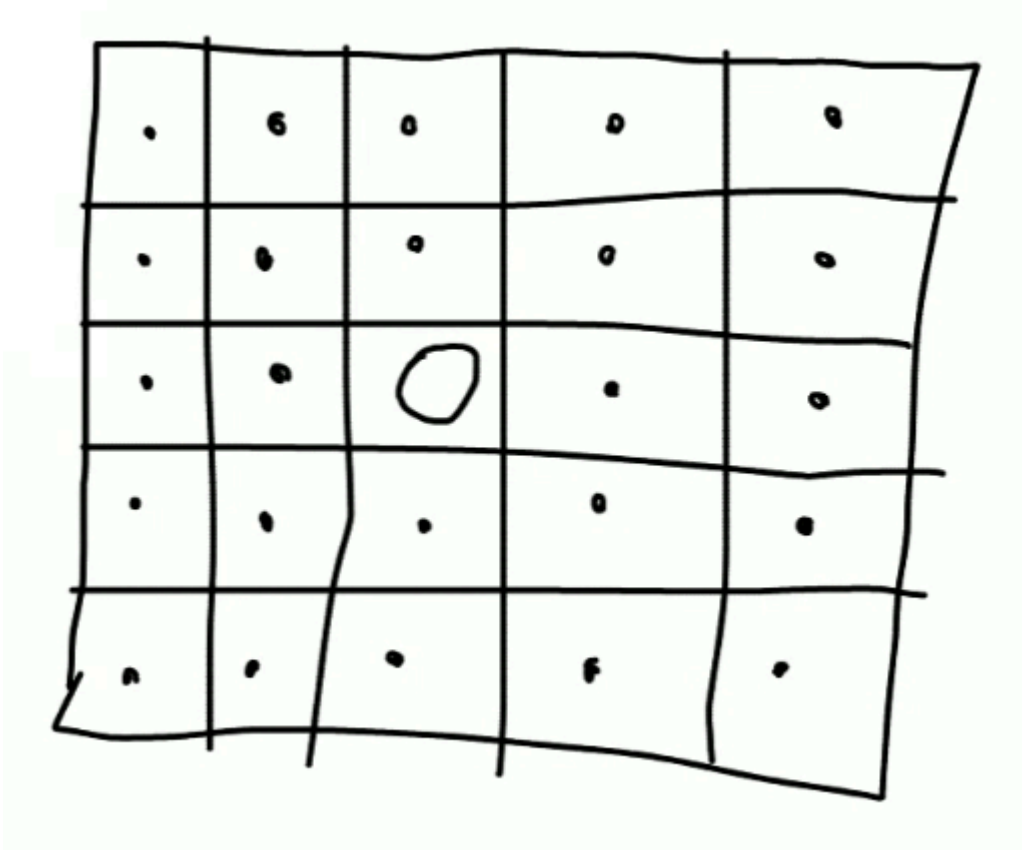
logo fizemos uma class do elevador com essas informações abstraídas, foi passado as informações padrão na classe para que não temos objetos com informações inconsistentes pois isso acabaria prejudicando bastante nosso sistema, foi apresentado também novos métodos de altera uma variável sem ter que altera diretamente no objeto para ter uma segurança maior garantindo que a variável não seja inconsistente por meio de funções que altere apenas quando permitido pelo sistema.

Foi passado um desafio ao final “desafio Chuck Norris” fazer o uso do sleep para percorrer a movimentação do elevador pelos andares.

```python
print('Olá mundo')

class Elevator:
    def __init__(self, maxFloor: int = 0, minFloor: int = 0,
currentFloor: int = 0, isDoorOpen: bool = False):
        self.__maxFloor = maxFloor
        self.__minFloor = minFloor
        self.__currentFloor = currentFloor
        self.__isDoorOpen = isDoorOpen

    # Propriedades para acessar os atributos privados
    @property
    def maxFloor(self):
        return self.__maxFloor

    @property
    def minFloor(self):
        return self.__minFloor

    @property
    def currentFloor(self):
        return self.__currentFloor

    @property
    def isDoorOpen(self):
```

```
        return self.__isDoorOpen

    # Setters para alterar os valores
    @maxFloor.setter
    def maxFloor(self, newMaxFloor: int):
        if 0 < newMaxFloor < 15:
            self.__maxFloor = newMaxFloor

    @minFloor.setter
    def minFloor(self, newMinFloor: int):
        if 0 < newMinFloor < 15:
            self.__minFloor = newMinFloor

    @currentFloor.setter
    def currentFloor(self, newCurrentFloor: int):
        if 0 < newCurrentFloor < 15:
            self.__currentFloor = newCurrentFloor

    @isDoorOpen.setter
    def isDoorOpen(self, doorState: bool):
        self.__isDoorOpen = doorState
    def move(self, newFloor: int):
        if not self.isDoorOpen:
            self.currentFloor = newFloor



serviceElevator = Elevator(maxFloor=10, isDoorOpen=True)
socialElevator = Elevator(maxFloor= 10)

serviceElevator.maxFloor = 12
serviceElevator.maxFloor = -1000
serviceElevator.isDoorOpen = False
serviceElevator.move(10)

serviceElevator.move(10)
serviceElevator.move(2)
serviceElevator.move(8)
serviceElevator.move(1)



print(serviceElevator.maxFloor)
```

```
print(serviceElevator.isDoorOpen)
print(serviceElevator.currentFloor)
A
```



```
class RobotCleaner:
    #matriz que vai representar o ambiente (com suas celulas e sugeira)
    #direcao
    #posicao atual dele (linha, coluna)
    #Carga de bateria Unidades X Unidades ex: carga inicial 15, detalhe, cada movimento consome 1 unidade de bateria
    #se carga zero, precisa parar X segundos para cada unidade de bateria recarregada
```

fazer o inicializador e métodos que de busca

desafio do elevador com o Sleep, bom acabei colocando um efeito sonoro tambem:

```
import time
import winsound  # windows apenas
print('Olá mundo')
class Elevator:
    def __init__(self, maxFloor: int = 0, minFloor: int = 0,
currentFloor: int = 0, isDoorOpen: bool = False):
        self.__maxFloor = maxFloor
        self.__minFloor = minFloor
        self.__currentFloor = currentFloor
        self.__isDoorOpen = isDoorOpen
    @property
    def maxFloor(self):
        return self.__maxFloor
    @property
    def minFloor(self):
        return self.__minFloor
    @property
    def currentFloor(self):
        return self.__currentFloor
    @property
    def isDoorOpen(self):
        return self.__isDoorOpen
    @maxFloor.setter
    def maxFloor(self, newMaxFloor: int):
        if 0 < newMaxFloor < 15:
            self.__maxFloor = newMaxFloor
    @minFloor.setter
    def minFloor(self, newMinFloor: int):
        if 0 < newMinFloor < 15:
            self.__minFloor = newMinFloor
```

```
    @currentFloor.setter
    def currentFloor(self, newCurrentFloor: int):
        if 0 <= newCurrentFloor <= self.maxFloor:
            self.__currentFloor = newCurrentFloor
    @isDoorOpen.setter
    def isDoorOpen(self, doorState: bool):
        self.__isDoorOpen = doorState
    def move(self, newFloor: int):
        if self.isDoorOpen:
            print(" Feche a porta antes de mover o elevador!")
            winsound.Beep(1000, 600)
            winsound.Beep(600, 800)
            return
        if newFloor > self.maxFloor or newFloor < self.minFloor:
            print(" Andar inválido!")
            return
        print(f"\nPlimPlomm\nSaindo do andar {self.currentFloor} para o
andar {newFloor}...\nPlimPlomm\n")
        if newFloor > self.currentFloor:  #up
            for andar in range(self.currentFloor + 1, newFloor + 1):
                time.sleep(1)
                winsound.Beep(800, 200)
                print(f"Elevador no andar {andar}")
                self.__currentFloor = andar
        elif newFloor < self.currentFloor:  #dowm
            for andar in range(self.currentFloor - 1, newFloor - 1,
-1):
                time.sleep(1)
                winsound.Beep(600, 200)
                print(f"Elevador no andar {andar}")
                self.__currentFloor = andar
        else:
            print("Elevador já está nesse andar!")
serviceElevator = Elevator(maxFloor=10, isDoorOpen=False)
socialElevator = Elevator(maxFloor=10)
serviceElevator.move(5)
serviceElevator.move(2)
serviceElevator.move(8)
serviceElevator.move(1)
print("Max Floor:", serviceElevator.maxFloor)
print("Door Open:", serviceElevator.isDoorOpen)
print("Current Floor:", serviceElevator.currentFloor)
```

Codigo do robo interativo :

```
import time

#precisa instalar a bibioteca= pip install keyboard
import keyboard

class RobotCleaner:
    def __init__(self, rows=15, cols=15, start=(0,0), battery=15,
recharge_time=1):
        self.__rows = rows
        self.__cols = cols
        self.__environment = [["." for _ in range(cols)] for _ in
range(rows)]
        self.__position = start
        self.__battery = battery
        self.__max_battery = battery
        self.__recharge_time = recharge_time
        self.__environment[start[0]][start[1]] = "R"

    @property
    def rows(self):
        return self.__rows
    @property
    def cols(self):
        return self.__cols
    @property
    def environment(self):
        return self.__environment
    @property
    def position(self):
        return self.__position
    @property
    def battery(self):
        return self.__battery
    @property
    def max_battery(self):
        return self.__max_battery
    @property
    def recharge_time(self):
        return self.__recharge_time

    @position.setter
    def position(self, new_pos):
```

```
        x, y = new_pos
        if 0 <= x < self.__rows and 0 <= y < self.__cols:
            self.__position = (x, y)

    @battery.setter
    def battery(self, new_battery):
        if 0 <= new_battery <= self.__max_battery:
            self.__battery = new_battery

    def show_environment(self):
        for row in self.__environment:
            print(" ".join(row))
        print("\n================================")
        print(f"bateria: {self.__battery}")
    def move(self, direction):
        if self.__battery <= 0:
            print("descarregado")
            self.recharge()
            return

        x, y = self.__position
        self.__environment[x][y] = " "
        if direction == "up" and x > 0:
            x -= 1
        elif direction == "down" and x < self.__rows - 1:
            x += 1
        elif direction == "left" and y > 0:
            y -= 1
        elif direction == "right" and y < self.__cols - 1:
            y += 1
        self.__position = (x, y)
        self.__environment[x][y] = "R"
        self.__battery -= 1

    def recharge(self):
        print("carregando")
        while self.__battery < self.__max_battery:
            time.sleep(self.__recharge_time)
            self.__battery += 1
            print(f"Bateria: {self.__battery}")
        print("carregado")

robot = RobotCleaner(start=(2,2), battery=55)
```

```
print("use as setas para mover \n")
robot.show_environment()
print("================================\n")
while True:
    if keyboard.is_pressed("up"):
        robot.move("up")
        robot.show_environment()
        time.sleep(0.2)
        print("================================\n")
    elif keyboard.is_pressed("down"):
        robot.move("down")
        robot.show_environment()
        time.sleep(0.2)
        print("================================\n")
    elif keyboard.is_pressed("left"):
        robot.move("left")
        robot.show_environment()
        time.sleep(0.2)
        print("================================\n")
    elif keyboard.is_pressed("right"):
        robot.move("right")
        robot.show_environment()
        time.sleep(0.2)
        print("================================\n")
    elif keyboard.is_pressed("esc"):
        print("xau...")
        break
```

eagora o codigo dele estatico:

```
import time
import random

class RobotCleaner:
    def __init__(self, rows=5, cols=5, start=(0,0), battery=15,
recharge_time=1):
        self.environment = [["." for _ in range(cols)] for _ in
range(rows)]
        self.rows = rows
```

```
        self.cols = cols
        self.position = start
        self.battery = battery
        self.max_battery = battery
        self.recharge_time = recharge_time
        self.environment[start[0]][start[1]] = "R"

    def show_environment(self):
        for row in self.environment:
            print(" ".join(row))
        print()

    def move(self, direction):
        if self.battery <= 0:
            print(" Bateria zerada! Recarregando...")
            self.recharge()
            return

        x, y = self.position
        self.environment[x][y] = " "
        if direction == "up" and x > 0:
            x -= 1
        elif direction == "down" and x < self.rows - 1:
            x += 1
        elif direction == "left" and y > 0:
            y -= 1
        elif direction == "right" and y < self.cols - 1:
            y += 1
        else:
            print("Movimento inválido!")
            return
        self.position = (x, y)
        self.environment[x][y] = "R"
        self.battery -= 1
        print("================================\n")
        print(f"Robô moveu para {self.position}, bateria:
{self.battery}")
        time.sleep(1)
        robot.show_environment()

    def recharge(self):
        robot.show_environment()
        print("================================\n")
```

```
        print(" Iniciando recarga...")
        for i in range(self.max_battery):
            time.sleep(self.recharge_time)
            self.battery += 1
            print(f"Bateria: {self.battery}")
        print(" Bateria carregada!")

    def clean(self):
        x, y = self.position
        if self.environment[x][y] == ".":
            self.environment[x][y] = "R"
            print("🧹 Sujeira limpa!")
        else:
            print("Nada para limpar aqui.")
robot = RobotCleaner(start=(2,2), battery=5)
robot.show_environment()
robot.move("up")
robot.move("up")
robot.move("left")
robot.move("left")
robot.move("down")
robot.move("down")
robot.move("right")
robot.move("down")
robot.move("left")
robot.move("down")
robot.move("down")
robot.move("right")
robot.move("up")
robot.move("right")
robot.move("down")
robot.move("right")
robot.move("up")
robot.move("right")
robot.move("down")
robot.move("right")
robot.move("up")
robot.move("up")
robot.move("up")
robot.move("up")
robot.move("up")
robot.move("right")
robot.move("down")
```

```
robot.move("down")
robot.move("down")
robot.move("down")
robot.move("down")

#robot.move("up")
#robot.move("left")
#robot.move("down")
#robot.move("right")
```