INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
CEARA

DEPARTAMENTO DE TELEMÁTICA
DISCIPLINA: PROGRAMAÇÃO ORIENTADA A OBJETO
LISTA EXERCICIO

ALUNO: Lucas Wagner Fernandes
Data: 29/09/2021

1ª Questão (10 Escores). Associe a cada item da 2ª coluna um valor que corresponde a um item da 1ª coluna.

a)	Permite que um objeto seja usado no lugar de outro.	(C)	Encapsulamento
b)	Define a representação de um objeto.	(H)	Mensagem
c)	Separação de interface e implementação que permite que usuários de objetos possam utilizá-los sem conhecer detalhes de seu código.	(I)	Herança
d)	Possui tamanho fixo.	(A)	Polimorfismo
e)	Instância de uma classe.	(F)	Dependência
f)	Forma de relacionamento entre classes onde objetos são instanciados código.	(J)	Lista
g)	Forma de relacionamento entre classes implementado por meio de coleções.	(B)	Classe
h)	Forma de chamar um comportamento de um objeto.	(E)	Objeto
i)	Reuso de código na formação de hierarquias de classes.	(G)	Composição
j)	Permite inserções e remoções.	(D)	Array

2ª Questão (10 Escores). Aplique V para as afirmações verdadeiras e F para as afirmações falsas.

a) Métodos construtores devem sempre ser explícitos. (F)

b) A classe **Professor** tem um relacionamento de agregação com a classe **Disciplina.** (V)

c) Quando uma classe possui como atributo uma referência para um objeto temos uma dependência. (V)

d) Membros de classes static existem mesmo quando nenhum objeto dessa classe exista. (V)

e) Um relacionamento ´***tem um***´ é implementado via herança. (F)

f) Uma classe **Funcionário** tem um relacionamento ´**é um**´ com a classe **Dependente.** (F)

g) Uma classe abstract pode ser instanciada. (F)

h) Relacionamentos TODO-PARTE são tipos de associações. (V)

i) Você implementa uma interface ao subscrever apropriada e concretamente todos os métodos definidos pela interface. (V)

j) Um método ***static*** não é capaz de acessar uma variável de instância. (F)

3ª Questão (40 Escores). Escreva exemplos de código Python onde seja possível identificar os seguintes conceitos de POO.

a) Herança;

```
class Animal:
    def __init__(self: object, specie: str, sound: str, color: str):
        self.specie = specie
        self.sound = sound
        self.color = color


class Dog(Animal):  # Dog herda de Animal
    def __init__(self: object, specie: str, sound: str, color: str, race: str):
        super().__init__(specie, sound, color)
        self.race = race
```

b) Encapsulamento;

```
class Adult:
    def __init__(self: object, name: str, age: int, salary: int):
        self.__name = name
        self.__age = age
        self.__salary = salary  # Atributos privados

    @property
    def salary(self) -> int:
        return self.__salary

    @salary.setter
    def salary(self, new_salary):
        raise ValueError

    def deposit(self, cash) -> int:
        return self.__salary + cash
```

c) Polimorfismo;

```
class Hello:
    def say(self):
        return "Hello!"

class Goodbye(Hello):
    def say(self):
        return "Goodbye!"


hello = Hello()
goodbye = Goodbye()
print(hello.say())
print(goodbye.say())  # Override no método say
```

d) Variáveis de Instância;

```
class Brazilian:
    def __init__(self: object, name: str, cpf: str, age: int):
        self.__name = name
```

```
        self.__cpf = cpf  # Variáveis de instância
        self.__age = age
```

e) Métodos construtores

Como existe uma ambiguidade, já que o método inicializador também geralmente é chamado de construtor:

```
class Car(object):
    def __init__(self, model: str, year: int):  # Inicializador
        self.model = model
        self.year = year


class Test:
    def __new__(cls):  # Construtor
        return cls()


teste = Test
car = Car("Opala", 1970)
```

f) Dependência

```
class Building:
    def __init__(self, name: str, floors: int, apts: int, address: str):
        self.__name = name
        self.__floors = floors
        self.__apts = apts
        self.__address = address

    @property
    def name(self):
        return self.__name


class Person:
    def __init__(self, name: str, local: type(Building)):  # Depende de Building
        self.__name = name
        self.__local = local

    def enterbuilding(self):
        return f"Entrando no {self.__local.name}!"


building = Building("Mirantes Passaré", 10, 30, "Passaré")
person = Person("Lucas", building)
print(person.enterbuilding())
```

g) Associação

```
# Associação entre as classes

class Jornalist:
    def __init__(self, name: str):
        self.__name = name
        self.__equipment = None

    @property
    def name(self) -> str:
        return self.__name
```

```
    @property
    def equipment(self):
        return self.__equipment

    @equipment.setter
    def equipment(self, equipment: object):
        self.__equipment = equipment


class Microphone:
    def __init__(self, color: str):
        self.__color = color

    def working(self):
        print("O microfone está funcionando!")

    def broken(self):
        print("O microfone está quebrado!")


jornalist = Jornalist("Lucas")
microphone = Microphone("Azul")
jornalist.material = microphone
jornalist.material.working()
jornalist.material.broken()
```

h) Relacionamento TODO-PARTE

```
# Exemplo de relacionamento TODO-PARTE

class Shopping:
    def __init__(self):
        self.__items = []

    def insert_items(self, product: object):
        self.__items.append(product)

    def list_products(self):
        for i in self.__items:
            print(i.name, i.value)

    def total(self) -> float:
        tl = 0
        for i in self.__items:
            tl += i.value

        return tl


class Product:
    def __init__(self, name: str, value: float):
        self.__name = name
        self.__value = value

    @property
    def name(self) -> str:
        return self.__name
```

```
    @property
    def value(self) -> float:
        return self.__value


shopping = Shopping()

product1 = Product("Camisa", 25.00)
product2 = Product("Sapato", 100.00)
product3 = Product("Mochila", 200.00)

shopping.insert_items(product1)
shopping.insert_items(product2)
shopping.insert_items(product3)

shopping.list_products()
print(shopping.total())
```

4ª Questão (20 Escores)

Escreva em Python uma classe Ponto que possui os atributos inteiros x e y. Escreva uma classe Reta que possui dois pontos a e b. Escreva os métodos construtores para a classe Ponto e para a Classe Reta. Escreva os métodos get e set para acessar e alterar os atributos da classe Ponto e da classe Reta. Escreva um método distancia que retorna um valor real da distancia entre os dois pontos da reta.

```
from math import sqrt


class Ponto:
    def __init__(self, x: int, y: int):
        self.__x = x
        self.__y = y

    @property
    def x(self) -> int:
        return self.__x

    @x.setter
    def x(self, new_x: int):
        self.__x = new_x

    @property
    def y(self) -> int:
        return self.__y

    @y.setter
    def y(self, new_y: int):
        self.__y = new_y


class Reta:
    def __init__(self, a: type(Ponto), b: type(Ponto)):
        self.__a = a
        self.__b = b

    @property
    def a(self) -> object:
```

```
        return self.__a

    @a.setter
    def a(self, new_a: object):
        self.__a = new_a

    @property
    def b(self) -> object:
        return self.__b

    @b.setter
    def b(self, new_b: object):
        self.__b = new_b

    def distancia(self) -> float:
        d = sqrt(((self.__b.x - self.__a.x)**2 + (self.__b.y - self.__a.y)**2))
        return d


def main():
    ponto1 = Ponto(2, 3)
    ponto2 = Ponto(4, 5)
    reta = Reta(ponto1, ponto2)
    print(f"Ponto 1({ponto1.x}, {ponto1.y})")
    print(f"Ponto 2({ponto2.x}, {ponto2.y})")
    print(f"Distância entre os 2 pontos da reta: {reta.distancia()}")


if __name__ == "__main__":
    main()
```