from abc import ABC, abstractmethod

# Interface para os produtos

class Computer(ABC):

@abstractmethod

def \_\_str\_\_(self):

pass

# Implementações concretas dos produtos

class PC(Computer):

def \_\_init\_\_(self, ram, hdd, cpu):

self.ram = ram

self.hdd = hdd

self.cpu = cpu

self.type = "PC"

def \_\_str\_\_(self):

return f"PC - RAM: {self.ram}GB, HDD: {self.hdd}GB, CPU: {self.cpu}GHz"

class Server(Computer):

def \_\_init\_\_(self, ram, hdd, cpu):

self.ram = ram

self.hdd = hdd

self.cpu = cpu

self.type = "Server"

def \_\_str\_\_(self):

return f"Server - RAM: {self.ram}GB, HDD: {self.hdd}GB, CPU: {self.cpu}GHz"

# Fábrica abstrata

class ComputerFactory(ABC):

@abstractmethod

def create\_computer(self, ram, hdd, cpu):

pass

# Implementações concretas da fábrica

class PCFactory(ComputerFactory):

def create\_computer(self, ram, hdd, cpu):

return PC(ram, hdd, cpu)

class ServerFactory(ComputerFactory):

def create\_computer(self, ram, hdd, cpu):

return Server(ram, hdd, cpu)

# Cliente utiliza a fábrica para criar instâncias do tipo abstrato

def main():

pc\_factory = PCFactory()

server\_factory = ServerFactory()

pc = pc\_factory.create\_computer(8, 500, 2.5)

server = server\_factory.create\_computer(16, 1000, 3.0)

print(pc)

print(server)

if \_\_name\_\_ == "\_\_main\_\_":

main()