Console Shop in Python

Author: Michał Deć

Nowy Sącz, 2024

1. Project scope

We will code simple console application in Python. This application will map computer shop. We will have clients that can order computer parts. Every client has his budget. There will be implemented basic logic. If client's budget is lower than computer part price, client order cannot be processed. In addition, every computer part has his quantity. If client wants to order more computer parts than we actually have, his order also cannot be processed.

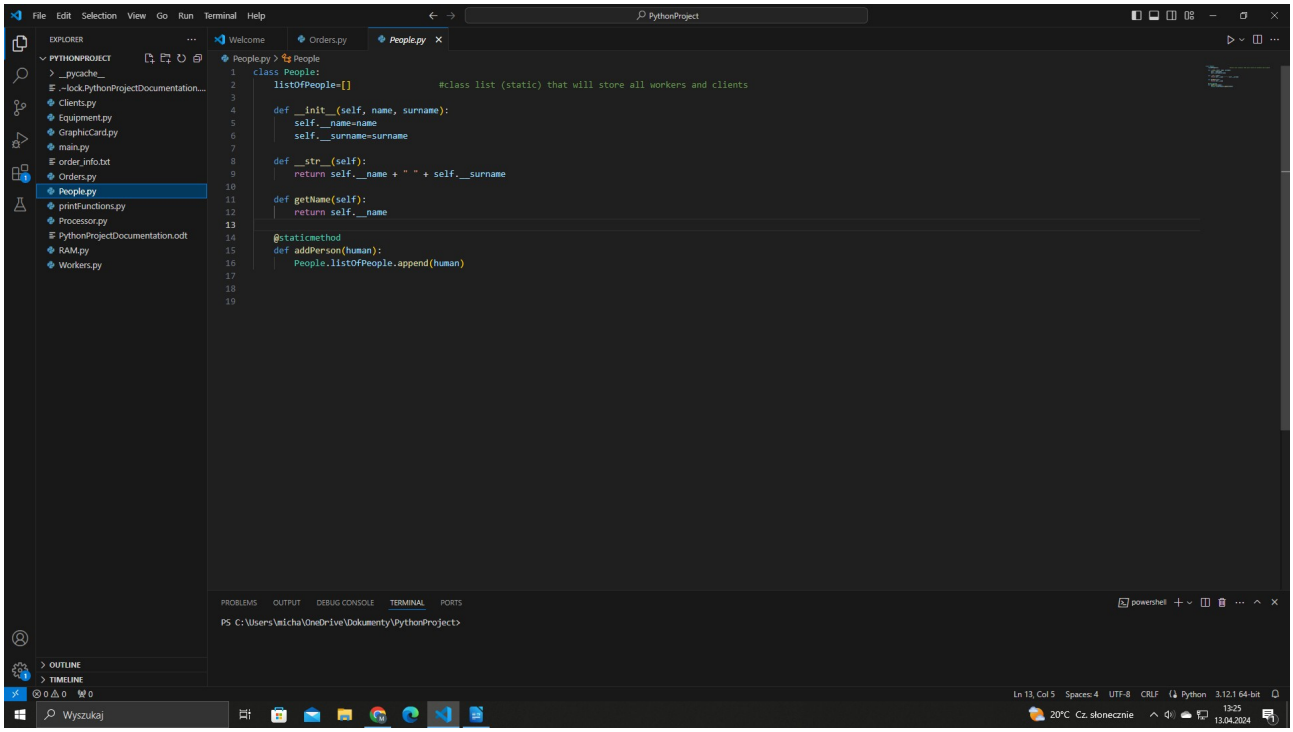
We will write this application with the use of Python 3.12.1 version. The main IDE for the project application will be Visual Studio Code. Application will be written with the OOP approach. We will break whole code into smaller modules, where classes are stored.

2. Program functionality

2.1 People module

2.1.1 Class People

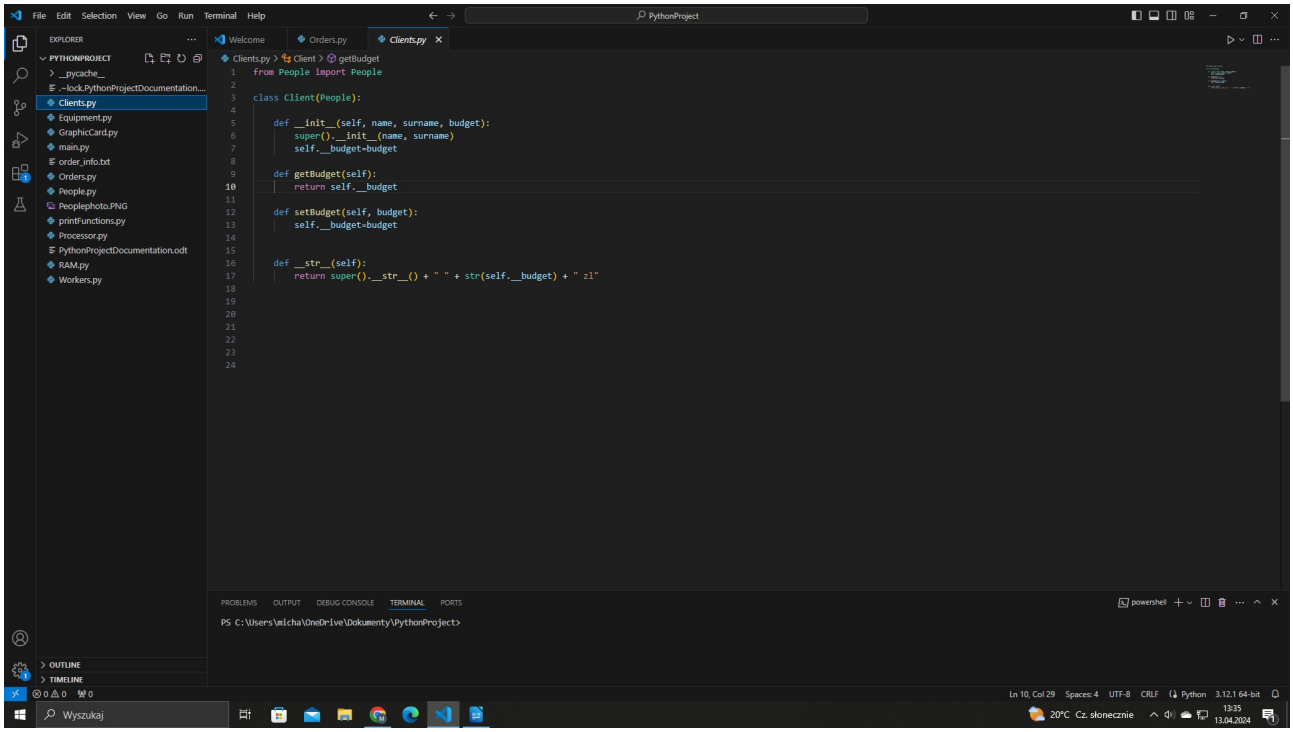
Inside People.py module we have class People. Every people has name and surname. Inside this class we have also static list, where we are going to store clients and workers. There is also method called addPerson, which allows for adding person to the list.



2.2 Clients module

2.2.1 Class Client

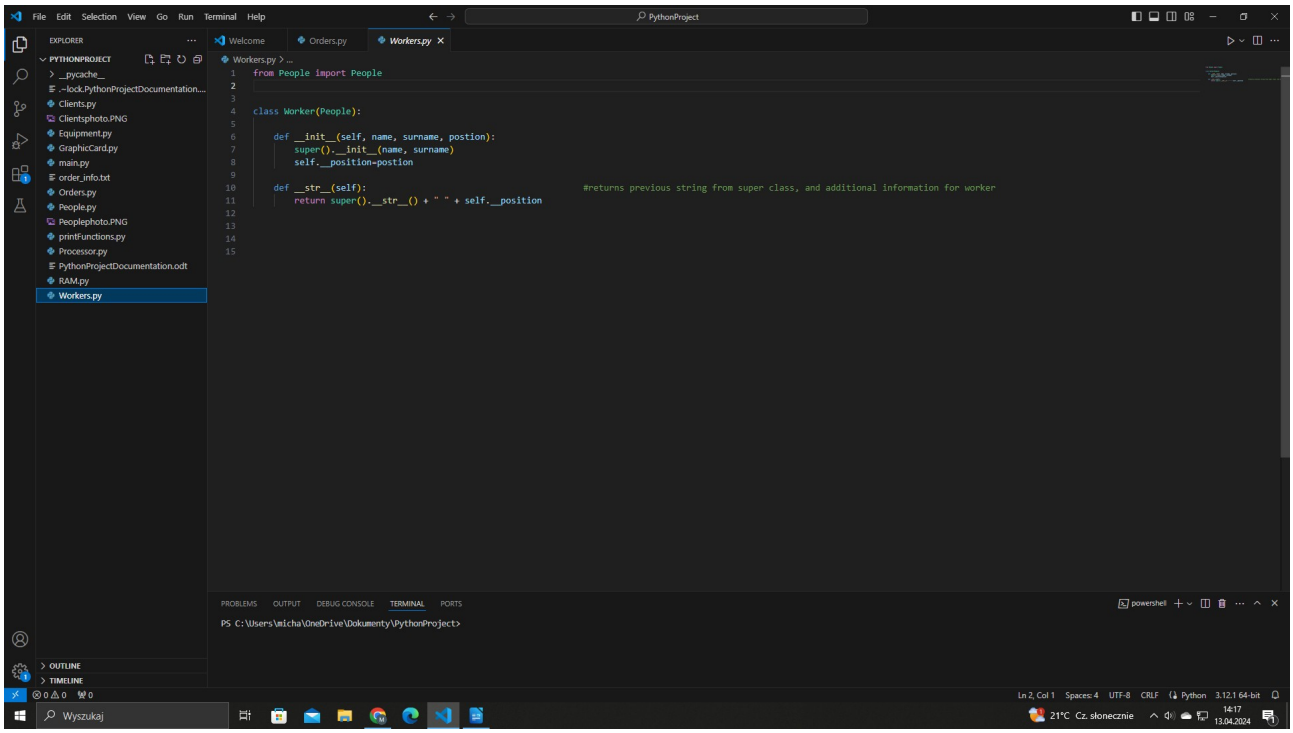
Inside Clients.py module we have defined class Client. Class Client inherits from class People. Every client has his budget. Inside Client class constructor we call constructor from super class, which in this case is People class. For the class Client we have also defined accessor and mutators for budget field. Budget is stored as private. We need get() and set() to manipulate this value.



2.3 Workers module

2.3.1 Class Worker

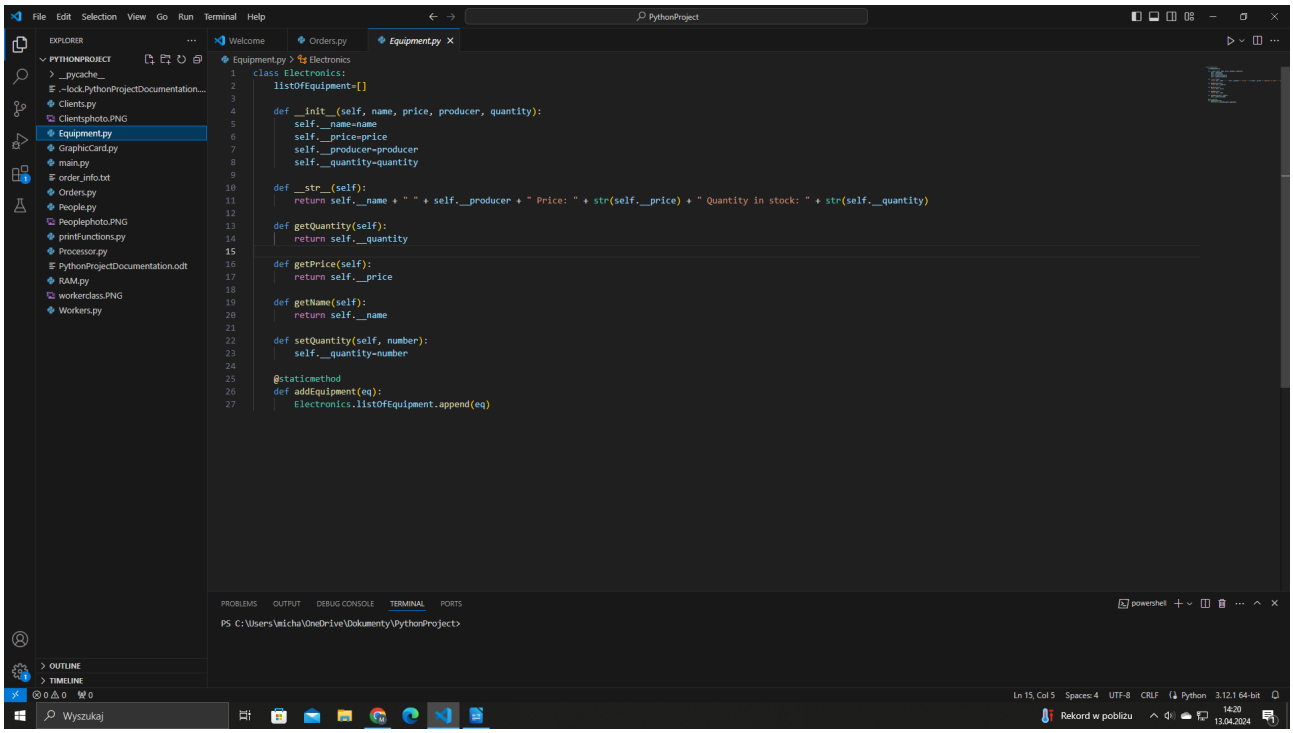
Inside Workers.py module we have defined class Worker. It inherits from class People. Inside class Worker we have constructor. It calls constructor from base class, which is class People. In addition, every worker has his role assigned, for example: „IT Support Desk”.



2.4 Equipment module

2.4.1 Class Electronics

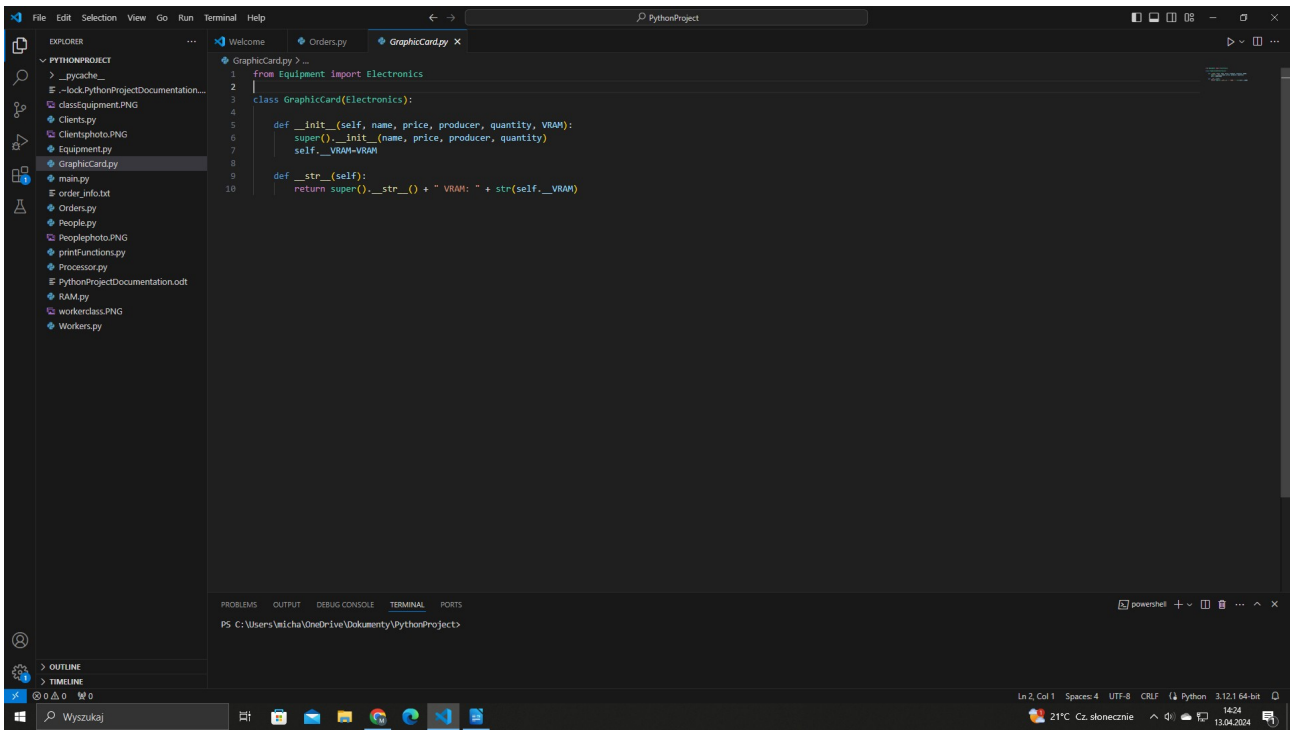
Inside Equipment.py module we have defined class Electronics. Every computer part has his name, value, producer, and quantity. Inside class Electronics we store static (class) list named listOfEquipment. We have also defined accessors and mutators for object's attributes.



2.5 GraphicCard module

2.5.1 Class GraphicCard

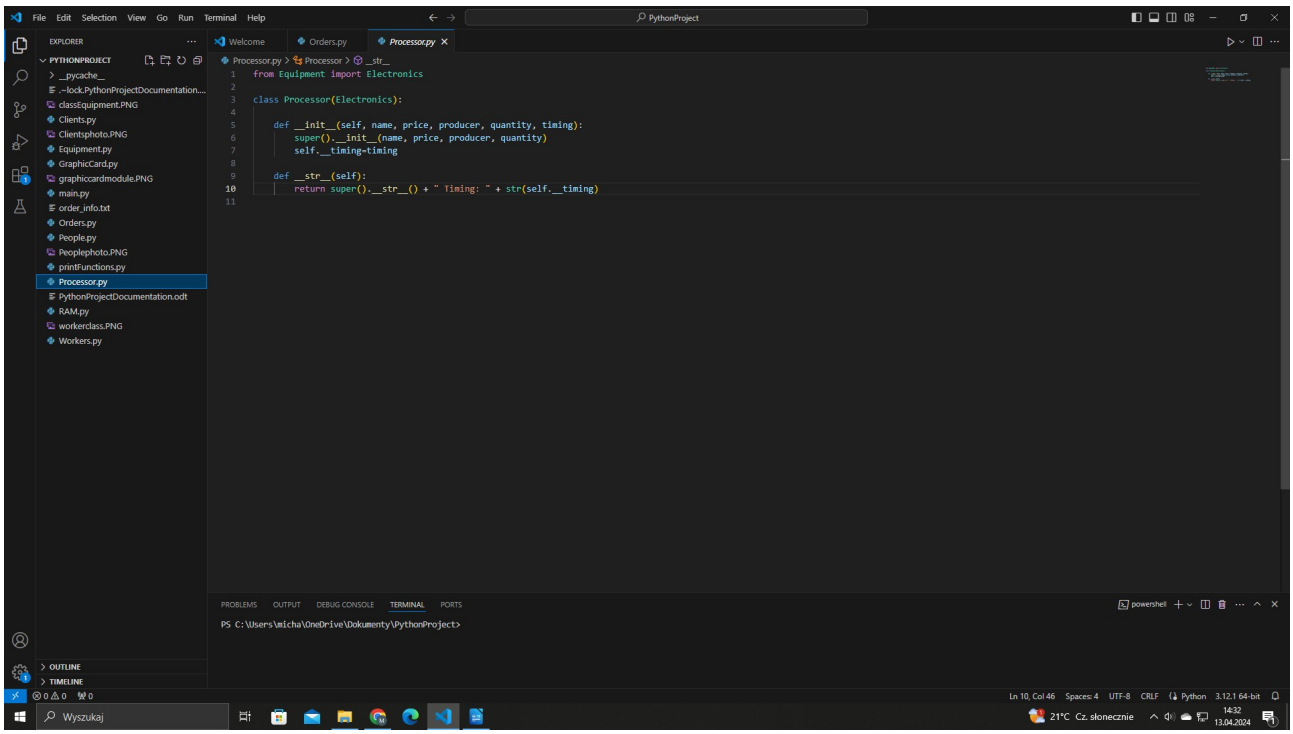
Inside GraphicCard.py module we have defined class GraphicCard. In inherits from class Electronics. In addition, every graphic card has defined field named VRAM.



2.6 Processor module

2.6.1 Class Processor

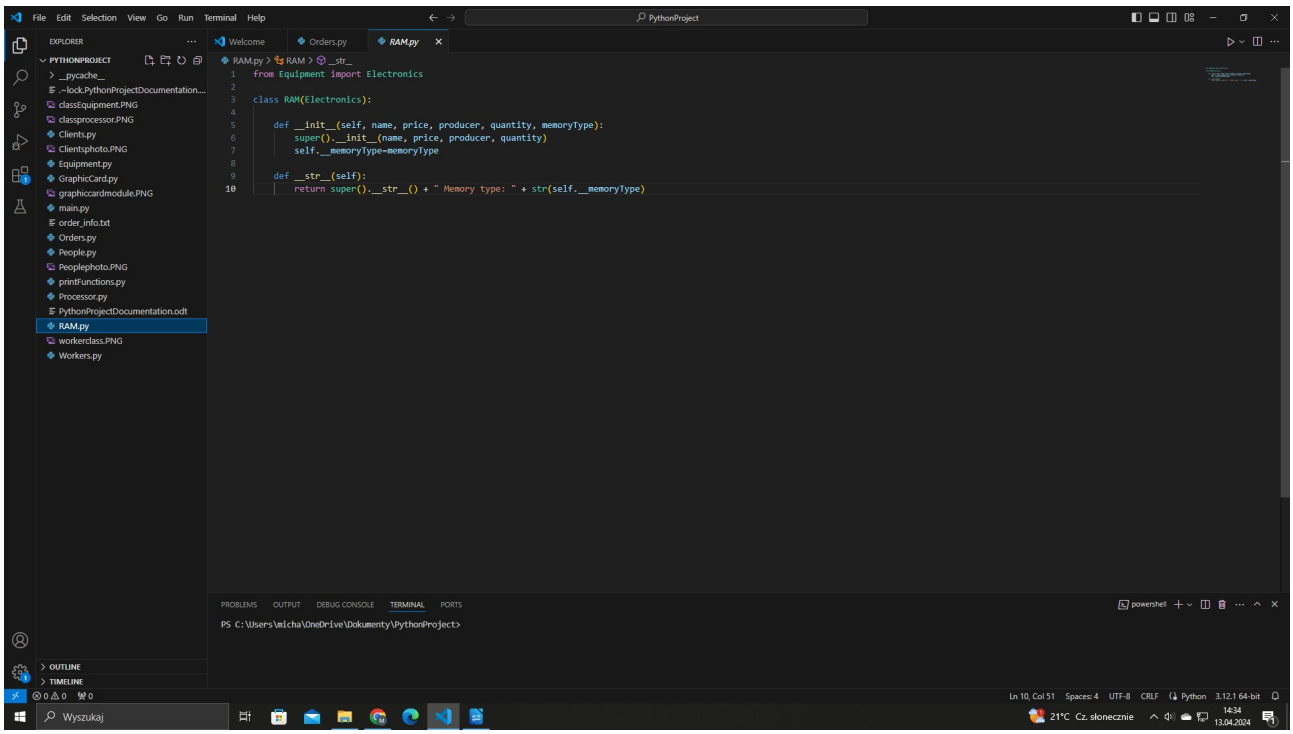
Inside Processor.py module we have defined class Processor. It inherits from class Electronics. In addition, every Processor has „timing” attribute, which describes CPU clock speed.



2.7 RAM module

2.7.1 Class RAM

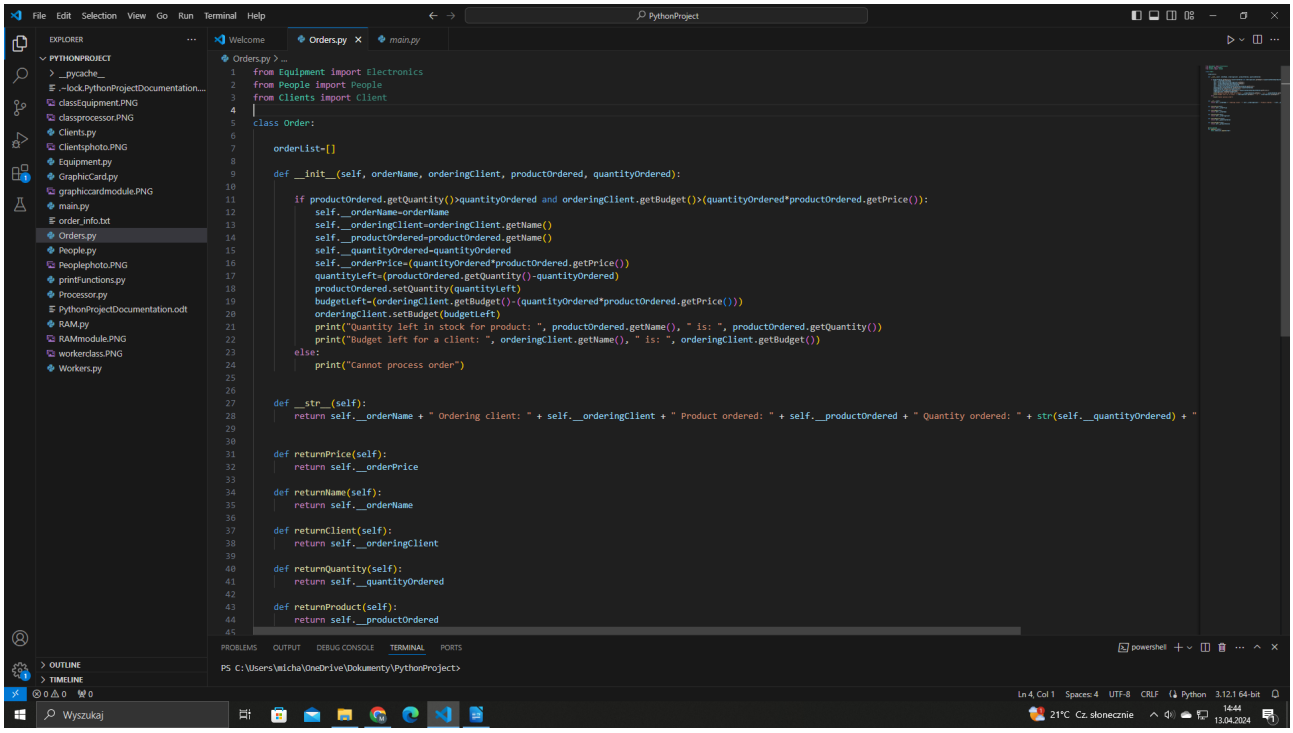
Inside RAM.py module we have defined class RAM. It inherits from class Electronics. In addition, every RAM component has field named „memoryType”, for example: DDR-4.



2.8 Orders module

2.8.1 Class Order

Inside Orders.py module we have defined class Order. We pass computer part and client as objects to the Order class contructor. Inside that constructor we have defined basic logic. If client's budget is greater than order price and we have more computer parts than client wants to order, we can process order. In any other condition, we cannot do it. If order is successfull, we also call mutators from Equipment and Client class. It subtracts client's budget by order price and also substracts equipment quantity by quantity ordered.



```python
from Equipment import Electronics
from People import People
from Clients import Client

class Order:

    orderList=[]

    def __init__(self, orderName, orderingClient, productOrdered, quantityOrdered):

        if productOrdered.getQuantity()>quantityOrdered and orderingClient.getBudget()>(quantityOrdered*productOrdered.getPrice()):
            self.__orderName=orderName
            self.__orderingClient=orderingClient.getName()
            self.__productOrdered=productOrdered.getName()
            self.__quantityOrdered=quantityOrdered
            self.__orderPrice=(quantityOrdered*productOrdered.getPrice())
            quantityLeft=(productOrdered.getQuantity()-quantityOrdered)
            productOrdered.setQuantity(quantityLeft)
            budgetLeft=(orderingClient.getBudget()-(quantityOrdered*productOrdered.getPrice()))
            orderingClient.setBudget(budgetLeft)
            print("Quantity left in stock for product: ", productOrdered.getName(), " is: ", productOrdered.getQuantity())
            print("Budget left for a client: ", orderingClient.getName(), " is: ", orderingClient.getBudget())
        else:
            print("Cannot process order")


    def __str__(self):
        return self.__orderName + " Ordering client: " + self.__orderingClient + " Product ordered: " + self.__productOrdered + " Quantity ordered: " + str(self.__quantityOrdered) + "


    def returnPrice(self):
        return self.__orderPrice

    def returnName(self):
        return self.__orderName

    def returnClient(self):
        return self.__orderingClient

    def returnQuantity(self):
        return self.__quantityOrdered

    def returnProduct(self):
        return self.__productOrdered
```

2.9 printFunctions module

2.9.1 printClients() function

It lists every client inside listOfPeople list from class People. It checks by instance, if current object is from Client class.

```
def printClients():
    for clients in People.listOfPeople:
        if isinstance(clients, Client):     #checks if current object has all attributes specified by Client class
            print(clients)
        else:
            continue
```

2.9.2 printWorkers() function

This function lists every worker inside listOfPeople list from class People. It checks by instance, if current object is from Worker class.

```
def printWorkers():
    for workers in People.listOfPeople:
        if isinstance(workers, Worker):
            print(workers)
        else:
            continue
```

2.9.3 printComponents() function

This function prints all computer components currently stored in list lsitOfEpuipment stored inside Equipment class.

```
def printComponents():
    for components in Electronics.listOfEquipment:
        print(components)
```

2.9.4 printOrder() function

This function prints every order currently stored in list orderList from Order class.

```
def printOrder():
    for orders in Order.orderList:
        print(orders)
```

2.9.5 ordersWithValueMoreThan5000() function

This function returns list with orders, which value is more then 5000. We have lambda, anonymous function, which lists every order with value greater than 5000 inside list orderList from class Order.

```
def ordersWithValueMoreThan5000():
    listWithOrdersWorthMoreThan5000 = list(filter(lambda totalPrice: totalPrice.returnPrice()>5000, Order.orderList))     #first it converts collection to another list,
    return listWithOrdersWorthMoreThan5000                                                                               #then lambda function takes only orders from Order List in Orc
                                                                                                                         #that are more expensive than 5000
```

2.9.6 printOrdersWithValueMoreThan5000 function()

Inside this function we call returned list from previous function. We list every order with value greater than 5000.

```
def printOrdersWithValueMoreThan5000():

    for orders in ordersWithValueMoreThan5000():
        print(orders)
```

2.9.7 saveOrdersToFile() function

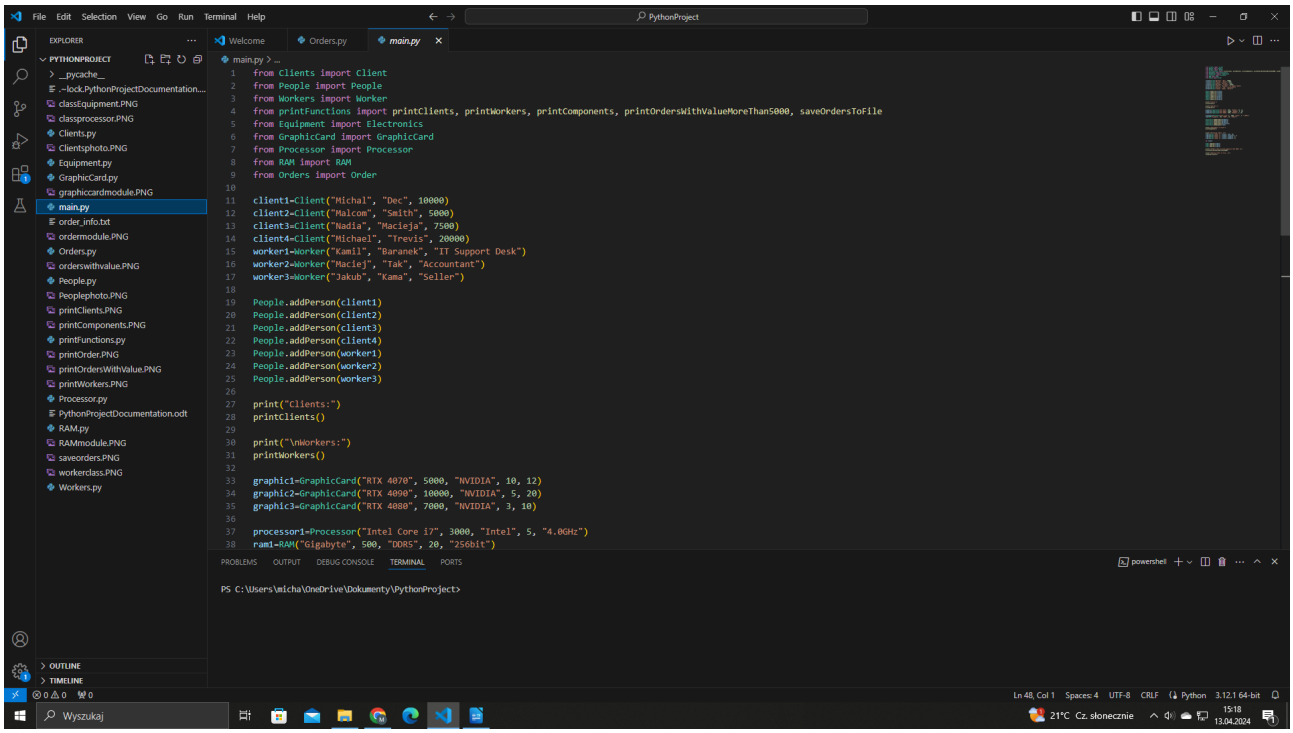
Inside this function we open buffer to text file named orders_info.txt. We list every order from list orderList from class Order. Then we save those orders to text file.

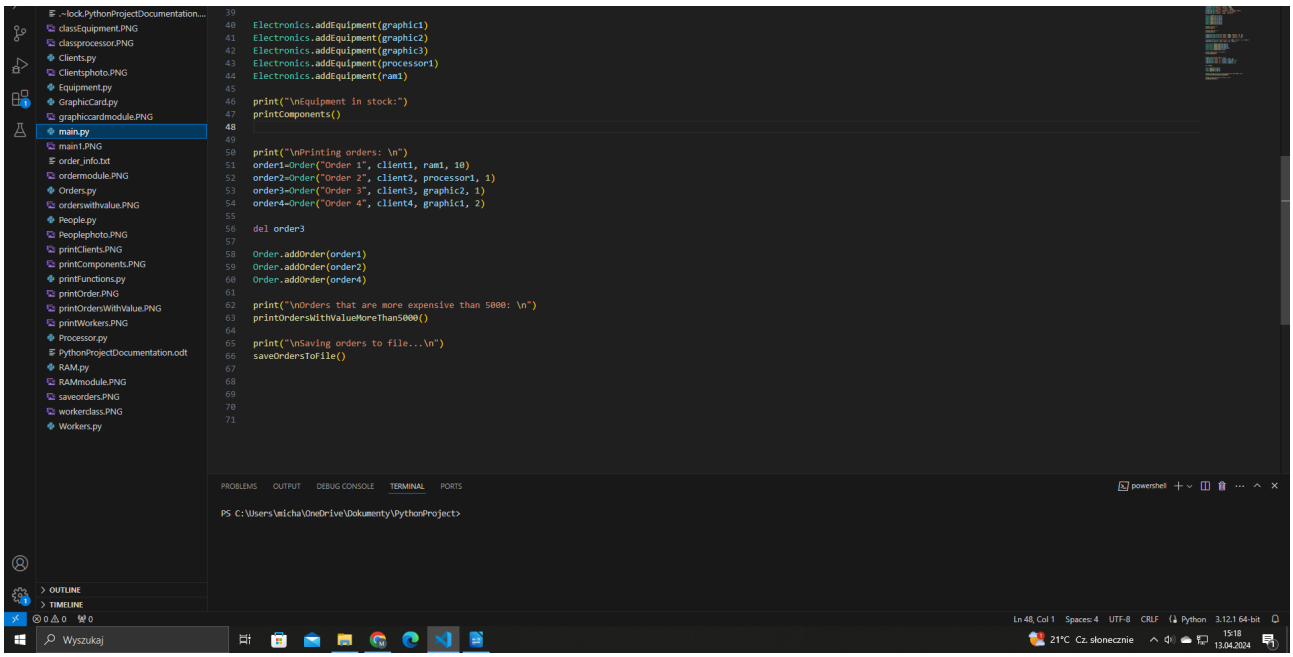
```
"""We define an object that we will use to save content from order list to a text file
    Then we iterate through every element in a order list, which is a static list"""

def saveOrdersToFile():
    with open("order_info.txt", "wt") as save:
        for element in Order.orderList:
            save.write(f"Product ordered: {element.returnProduct()} Quantity ordered: {element.returnQuantity()} Ordering client: {element.returnClient()} Order price: {element.returnP
```

2.10 main module

Inside main module we call all others modules and functions. Inside this main module we carry out all operations. We create objects, we add them to lists, and we process orders. We also save all orders to local file named orders_info.txt




.~lock.PythonProjectDocumentation....
classEquipment.PNG
classprocessor.PNG
Clients.py
Clientsphoto.PNG
Equipment.py
GraphicCard.py
graphiccardmodule.PNG
main.py
main1.PNG
order_info.txt
ordermodule.PNG
Orders.py
orderswithvalue.PNG
People.py
Peoplephoto.PNG
printClients.PNG
printComponents.PNG
printFunctions.py
printOrder.PNG
printOrdersWithValue.PNG
printWorkers.PNG
Processor.py
PythonProjectDocumentation.odt
RAM.py
RAMmodule.PNG
saveorders.PNG
workerclass.PNG
Workers.py
Electronics.addEquipment(graphic1)
Electronics.addEquipment(graphic2)
Electronics.addEquipment(graphic3)
Electronics.addEquipment(processor1)
Electronics.addEquipment(ram1)
print("\nEquipment in stock:")
printComponents()
print("\nPrinting orders: \n")
order1=Order("Order 1", client1, ram1, 10)
order2=Order("Order 2", client2, processor1, 1)
order3=Order("Order 3", client3, graphic2, 1)
order4=Order("Order 4", client4, graphic1, 2)
del order3
Order.addOrder(order1)
Order.addOrder(order2)
Order.addOrder(order4)
print("\nOrders that are more expensive than 5000: \n")
printOrdersWithValueMoreThan5000()
print("\nSaving orders to file...\n")
saveOrdersToFile()
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
powershell
PS C:\Users\micha\OneDrive\Dokumenty\PythonProject>
OUTLINE
TIMELINE
Ln 48, Col 1
Spaces: 4
UTF-8
CRLF
Python
3.12.1 64-bit
Wyszukaj
21°C Cz. słonecznie
15:18
13.04.2024

3. Conclusion

We created simple console application in Python, which maps computer shop. We could improve this application. For example we could create control panel, which allows for creating new clients, computer components, and order processing. Everything would be automated. In addition, we could create database server where we could store all data. We wouldn’t have to use lists, beacuse we could store this information inside database server, for example MariaDB.

Main goal of this application was to present basic logic behind OOP approach. We have classes, inheritance, composition, and overriding. These are basic concepts of Object Oriented Proggaming.

4. Materials and resources used for the need of this report

1) Python: Python version 3.12.1 - https://www.python.org/downloads/
2) IDE: Visual Studio Code - https://code.visualstudio.com/

List of contents