Lab: Design Patterns

Problems for in-class lab for the Python OOP Course @SoftUni.

1. Abstract Factory

We are going to create an **AbstractFactory** class that will have methods for creating **chairs**, **sofas**, and **tables**:

```
from abc import ABC, abstractmethod

class AbstractFactory(ABC):
    @abstractmethod
    def create_chair(self):
        raise NotImplementedError()

    @abstractmethod
    def create_sofa(self):
        raise NotImplementedError()

    @abstractmethod
    def create_table(self):
        raise NotImplementedError()
```

Then we are going to create the **Chair**, **Sofa** and **Table** classes:

```
class Chair:
    def __init__(self, name):
        self._name = name

    def __str__(self):
        return self._name

class Sofa:
    def __init__(self, name):
        self._name = name

    def __str__(self):
        return self._name

class Table:
    def __init__(self, name):
        self._name = name

    def __str__(self):
        return self._name
```

Finally, we are going to create **three** different factories that **will inherit** from the **`AbstractFactory`** class: **`VictorianFactory`**, **`ModernFactory`**, **`FuturisticFactory`**

```
class VictorianFactory(AbstractFactory):
    def create_chair(self):
        return Chair('victorian chair')

    def create_sofa(self):
        return Sofa('victorian sofa')

    def create_table(self):
        return Table('victorian table')

class ModernFactory(AbstractFactory):
    def create_chair(self):
        return Chair('modern chair')

    def create_sofa(self):
        return Sofa('modern sofa')

    def create_table(self):
        return Table('modern table')

class FuturisticFactory(AbstractFactory):
    def create_chair(self):
        return Chair('futuristic chair')

    def create_sofa(self):
        return Sofa('futuristic sofa')

    def create_table(self):
        return Table('futuristic table')
```

2. Composite

We are going to create a class called **`Component`** which will have **move** and **delete** methods:

```
class Component:
    def __init__(self, name):
        self.name = name
        self.parent = None

    def move(self, new_path):
        new_folder = get_path(new_path)
        del self.parent.children[self.name]
        new_folder.children[self.name] = self
        self.parent = new_folder

    def delete(self):
        del self.parent.children[self.name]
```

Create two more classes: **Folder** and **File**, which **inherit** from **Component**. The **Folder** class should have an **add_child** method:

```
class Folder(Component):
    def __init__(self, name):
        super().__init__(name)
        self.children = {}

    def add_child(self, child):
        self.parent = self
        self.children[child.name] = child


class File(Component):
    def __init__(self, name, contents):
        super().__init__(name)
        self.contents = contents
```

Create a **root folder** and implement the method **get_path**:

```
root = Folder('')

def get_path(path):
    names = path.split('/')[1:]
    node = root
    for name in names:
        node = node.children[name]
    return node
```

3. Command

Create class **Window** and class **Document**. The Window class will have an **exit** method. The Document class will have a **save** method:

```
import sys

class Window:
    def exit(self):
        sys.exit(0)

class Document:
    def __init__(self, filename):
        self.filename = filename
        self.contents = "This file cannot be modified"

    def save(self):
        with open(self.filename, 'w') as file:
            file.write(self.contents)
```

Next, we will create the classes **ToolbarDocument**, **MenuItem**, and **KeyboardShortcut**. The **ToolbarDocument** and the **MenuItem** will have a method **click**. The **KeyboardShortcut** should have a method called **keypress**.

```
class ToolbarDocument:
    def __init__(self, name, iconname):
        self.name = name
        self.iconname = iconname

    def click(self):
        self.command.execute()

class MenuItem:
    def __init__(self, menu_name, item_name):
        self.menu = menu_name
        self.item = item_name

    def click(self):
        self.command.execute()

class KeyboardShortcut:
    def __init__(self, key, modifier):
        self.key = key
        self.modifier = modifier

    def keypress(self):
        self.command.execute()
```

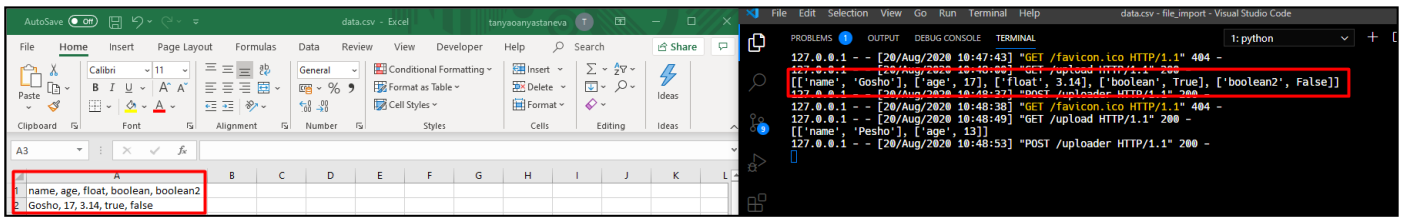
Finally, we will create two **command classes SaveCommand** and **ExitCommand** which will have the **execute** methods implemented in them:

```
class SaveCommand:
    def __init__(self, document):
        self.document = document

    def execute(self):
        self.document.save()

class ExitCommand:
    def __init__(self, window):
        self.window = window

    def execute(self):
        self.window.exit()
```

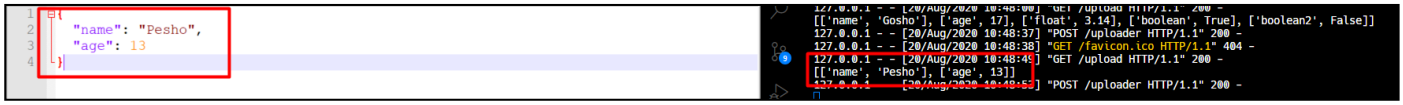
4. BONUS: File Uploader

Create a simple **server** (**flask**) for file uploads of type **`json`** and **`csv`**. Render a **simple HTML form** and on **submit**:

- If the **format** of the file is **correct**, a message should be displayed in the browser: **`"File uploaded successfully"`**, and information about the file should be **printed on the console**
    - **CSV** Example:



    - **JSON** Example:



- Otherwise, another message should be displayed: **`"Invalid content type for import"`**

Note: Use the design patterns you have learned to build your project

Useful Links

- Flask **documentation**: https://flask.palletsprojects.com/en/1.1.x/api/
- Example **solution**: https://github.com/Minkov/python-oop-2020-06/tree/demos/file_import

Follow us: