Design Pattern

When we want to limit the number of object to be created when someone want to create class object we use **__new__** class to achieve desired criteria for that.

Singleton

--new-- (cls):

- It is called before the constructors for e.g. **__init__(self)** as we know very well about **self** which is nothing but instance reference of class.
- Surprisingly **__new__** generates that object reference so to limit the number of generated we have to override **__new__** method.

```
class SingleTone:
    """SingleTone class"""

    def __new__(cls):
        """Responsible for creation of new object"""
        if not hasattr(cls, "instance"):
            cls.instance = super().__new__(cls)
        return cls.instance

s1 = SingleTone()
s2 = SingleTone()
print(id(s1),id(s2))
```

If we print the ids of s1 and S2 we will get **2127268224784** and **2127268224784** (changes when run again but it will be same) which are same.

By calling `().__new__(cls)` , we are inheriting the `__new__(cls)` and also overriding it.

We will get same object reference even if we run it thousands time.
it is advisable to use `._instance` or `.__instance` to make it protected or private respectively.

Form Factory

Form Factory creates forms:

1. Using direct method

```
class Form:
    html = ""
    def get_html(self):
        return self.html

class Text(Form):
    html = '<input type = "text" value= "Dummy Data Text">' # defining static
variable html when function is called

class Email(Form):
    html = '<input type = "text" value= "Dummy Data Email">'#

class Password(Form):
    html = '<input type = "text" value= "Dummy Data Password">'


# Methode 1 calling methodes directly when condition is satiesfied
class FormFactory():
    def create_form(self, inp):
        inp_data = inp.title()
        if inp_data == 'Text':
            obj = Text() # after this line html = '<input type = "text" value=
"Dummy Data Text">'
        elif inp_data == 'Email':
            obj = Email()
        elif inp_data == 'Password':
            obj = Password()
        else:
            print("Invalid class name")
        if obj:
            return obj


ff = FormFactory()
res = ff.create_form('email') # html =  value chnage to email tag
print(res.get_html()) # prints html value
```

output:

When we **email** as an argument in create_form

```
D:\Python_playground\Practice\Design
Pattern>C:/Users/Sharique/AppData/Local/Programs/Python/Python39/python.exe
"d:/Python_playground/Practice/Design Pattern/Design pattern.py"
<input type = "email" value= "Dummy Data Email">
```

2. Using global method

```
class Form:
    html = ""
    def get_html(self):
        return self.html

class Text(Form):
    html = '<input type = "text" value= "Dummy Data Text">'

class Email(Form):
    html = '<input type = "email" value= "Dummy Data Email">'

class Password(Form):
    html = '<input type = "password" value= "Dummy Data Password">'


# Methode 2 using global methode
class FormFactory():
    def create_form(self, inp):
        l = ["Text","Password",'Email']
        inp = inp.title()
        if inp.title() in l:
            class_name = globals()[inp] # object creation
            obj = class_name()
            print(obj.get_html()) # calling get_html using class object
        else:
            print("Invalid Input")


ff = FormFactory()
ff.create_form('Password')
```

output

```
D:\Python_playground\Practice\Design
Pattern>C:/Users/Sharique/AppData/Local/Programs/Python/Python39/python.exe
"d:/Python_playground/Practice/Design Pattern/Design pattern.py"
<input type = "password" value= "Dummy Data Password">
```

3. Using eval method

```
class Form:
    html = ""
    def get_html(self):
        return self.html

class Text(Form):
    html = '<input type = "text" value= "Dummy Data Text">'

class Email(Form):
    html = '<input type = "email" value= "Dummy Data Email">'

class Password(Form):
    html = '<input type = "password" value= "Dummy Data Password">'


# Methode 2 using global methode
class FormFactory():
    def create_form(self, inp):
        l = ["Text","Password",'Email']
        inp = inp.title()
        if inp in l:
           obj = eval(inp)() # creating objects
           return obj.get_html() # calling get_html()
        else:
            print("Invalid Input")


ff = FormFactory()
l = ["Text","Password",'Email']
for inp in l:
    res = ff.create_form(inp)
    print(res)
```

Output

```
D:\Python_playground\Practice\Design
Pattern>C:/Users/Sharique/AppData/Local/Programs/Python/Python39/python.exe
"d:/Python_playground/Practice/Design Pattern/Design pattern.py"
<input type = "text" value= "Dummy Data Text">
<input type = "password" value= "Dummy Data Password">
<input type = "email" value= "Dummy Data Email">
```

Another example of factory:

Creating computer class and getting configuration:

```
class Computer:
    # ram = None
    # rom =  None
    # graphics = None
    def __init__(self, ram, rom, graphics):
        self.ram = ram
        self.rom = rom
        self.graphics = graphics


    def get_configuration(self):
        print(f"""
        RAM:- {self.ram}
        ROM:- {self.rom}
        Graphics:- {self.graphics}
        """)


class Desktop(Computer):
    # ram = "8 GB"
    # rom =  "500 GB"
    # graphics = "2 GB"
    pass


class Laptop(Computer):
    # ram = "16 GB"
    # rom =  "1 TB"
    # graphics = "4 GB"
    pass


class ComputerFactory:
    def create_computer(self, comp_type,ram,rom,graphics):
        if comp_type in ['laptop', 'desktop']:
            resp = comp_type.title()
        else:
            raise ValueError(f"No any class present named :- {comp_type}")

        if resp:
            class_name = eval(resp)
            obj = class_name(ram,rom,graphics)
            return obj


cf = ComputerFactory()
obj_1 = cf.create_computer("laptop","4 GB","500 GB", "2 GB")
obj_2 = cf.create_computer("laptop","8 GB","1 TB", "4 GB")
obj_1.get_configuration()
obj_2.get_configuration()
```

References:

1. Refactoring.guru
2. Tutorial Point
3. GitHub: Arav_tech