Série TP n°6 2022-2023	Module	MTI Méthd. et tech d'implement.
	Filière	Master GSI 1 ère Année
Chapitre 3: Design patterns	ملاحظة :	

- **Objectifs** أهداف: Design patterns (Factory method, Abstract Factory Method)
- **Données** بيانات:
- **Outils** أدوات : Python

1 Partie TP

1.1 Partie 1

1.1.1 Sans Design pattern

1- Exécuter le programme suivant :

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#http://python-3-patterns-idioms-test.readthedocs.io/en/latest/Factory.html
# A simple static factory method.
import sys
class Circle(Shape):
    def draw(self): print("Circle.draw")
    def erase(self): print("Circle.erase")

class Square(Shape):
    def draw(self): print("Square.draw")
    def erase(self): print("Square.erase")
if __name__ == "__main__":
  for type in ("Circle", "Square" ,"Circle", "Square"):
    if type == "Circle":
        shape = Circle()
    elif type == "Square":
        shape = Square()
    else:
        print "Bad shape creation: " + type
        sys.exit()
    shape.draw()
    shape.erase()
```

2- Modifier le code pour ajouter une forme triangle et rectangle.

1.1.2 Simple Factory method

3- on veut encapsuler l'operation d'instanciation des objets, en utilsant une classe ShapeFactory

```
class ShapeFactory:
    @staticmethod
    def createShape(type):
        if   type == "Circle":  return Circle()
        elif type == "Square":  return Square()
        else:
            print "Bad shape creation: " + type
            sys.exit()
if __name__ == "__main__":
```

```
for type in ("Circle", "Square" ,"Circle", "Square"):
  shape = ShapeFactory.createShape(type)
  shape.draw()
  shape.erase()
```

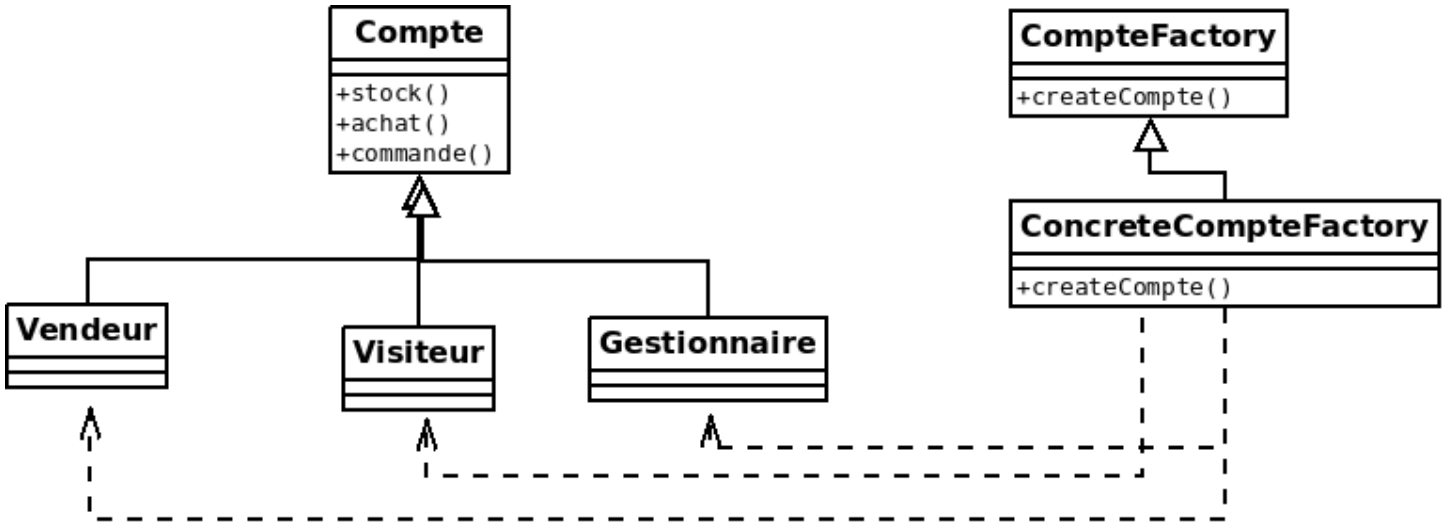
2- Modifier le code pour ajouter les formes Triangle et rectangle.

1.1.3 Factory method

3- On veut créer une Factory (ShapeFactory_SCT) spécialisée qui ne fabrique que les carrés, les cercles, et les triangles. Une autre Factory (ShapeFactory_SCR) qui fabrique que les carrés et les cercles et les rectangles.

Application : On veut une application de gestion de stock, notre client exige d'avoir trois types de comptes :

- un compte gestionnaire, pour gérer le stock, faire des achats et des commandes, voir la recette.
- un compte vendeur, pour permettre aux vendeurs de vendre, faire des achats et des commandes.
- Un compte visiteur pour permettre aux clients de consulter le stock, faire des achats et des commandes.
- Proposer un design pattern, en inspirant du diagramme de classe.
- Implémenter les classes de base et la classe de création.

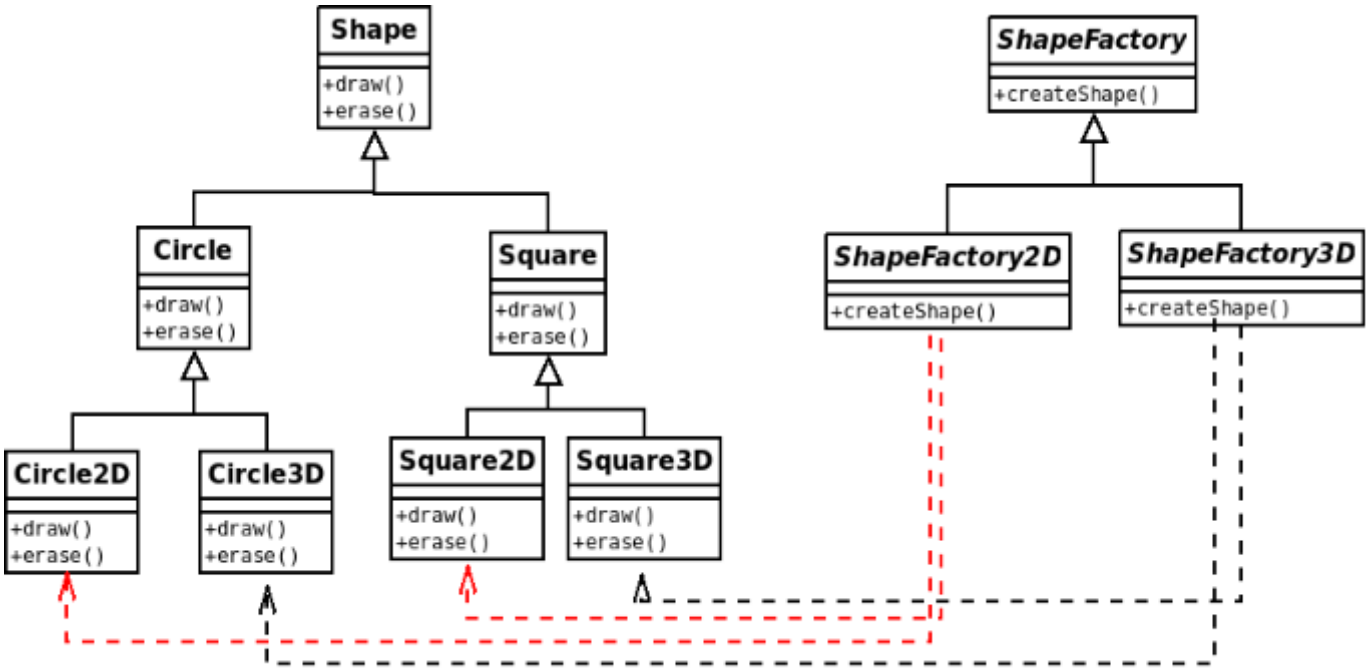


1.2 Partie 2 Abstract Factory

On reprends les classes de la première partie.

On veut Créer des formes 2D et des formes 3D, selon le diagramme suivant.

1. créer les sous classes square2D, Square3D, circle2D, circle3D.
2. Créer l'Usine (Factory) abstraite, ShapeFactory
3. Créer les usines concrètes ShapeFactory2D, et ShapeFactory3D



4. Le code suivant illustre juste des exemples.

1.3 Partie 3 Singleton

Tester le code suivant :

```
class  Singleton (object):
    instance = None   # Attribut statique de classe
    def __new__(cls):
        "méthode de construction standard Python"
        if cls.instance is None:
            cls.instance = object.__new__(cls)
        return cls.instance
# Utilisation
monSingleton1 =  Singleton()
monSingleton2 =  Singleton()
# monSingleton1 et monSingleton2 renvoient à la même instance
assert monSingleton1 is monSingleton2
print monSingleton1, monSingleton2
```

1. Proposer un classe qui permet d"ouvrir un fichier à puis en écriture, Il faut s'assurer que l'accès au fichier est unique
2. Proposer un classe qui permet de se connecter à une base de données puis la fermer, Il faut s'assurer que l'accès à la base de données est unique
3. Améliorer la solution en permettant l'accès simultané au fichier /base de données pour un nombre limité

1.4 Partie 4 Builder Pattern

Soit la classe story qui simule les stories de Facebook, instegramme et Youtube.

Une Story est composée de photo, video, effets spéciaux, décorations, music, mais elle ne contient pas tous ces composants en même temps. Donc on peut avoir plusieurs configurations.

```
class Story:
    """
    Represent the complex object under construction.
```

```
    """
    def __init__(self,):
        self.parts = []
    def add(self, part):
        self.parts.append(part)
    def show(self,):
            print(self.parts)
```

1. On veut créer une classe qui permet de créer les différent composants d'une story.

```
class Builder():
    """
    Specify an abstract interface for creating parts of a Product
    object.
    """
    def __init__(self):
        self.story = Story()

    def _build_image(self):
        pass

    def _build_music(self):
        pass

    def _build_effet(self):
        pass
    def _build_video(self):
        pass
```

2. On veut créer une classe qui permet de configuer la création des stories en :

   - story basic (juste une photo)
   - story avec music
   - story video avec effets speciaux

```
class StoryDirector:
    """
    Construct an object using the Builder interface.
    """
    def __init__(self):
        self._builder = None
    def construct(self, builder):
        self._builder = builder
    def basic_story(self,):
        print("*** create basic story ***")
        self._builder._build_image()
    def photo_music_story(self,):
        print("*** create photo + music story ***")
        self._builder._build_image()
        self._builder._build_music()
        self._builder._build_effet()

    def video_story(self,):
        print("*** create video story ***")
```

```
        self._builder._build_video()
        self._builder._build_effet()
```

3. On veut créer une classe qui permet de créer les composants spéciaux pour FaceBook et Instegramme

```
class FacebookConcreteBuilder(Builder):
    """
    Construct and assemble parts of the product by implementing the
    Builder interface.
    Define and keep track of the representation it creates.
    Provide an interface for retrieving the product.
    """

    def _build_image(self):
        self.story.add("image facebook")

    def _build_music(self):
        self.story.add("music facebook")


    def _build_effet(self):
        self.story.add("effect facebook")
    def _build_video(self):
        self.story.add("video facebook")

class InstagramConcreteBuilder(Builder):
    """
    Construct and assemble parts of the product by implementing the
    Builder interface.
    Define and keep track of the representation it creates.
    Provide an interface for retrieving the product.
    """

    def _build_image(self):
        self.story.add("image Instagram")

    def _build_music(self):
        self.story.add("music Instagram")


    def _build_effet(self):
        self.story.add("effect Instagram")
    def _build_video(self):
        self.story.add("video Instagram")
```

4. tester l'appel de la fonction Main

```
def main():
    concrete_builder = FacebookConcreteBuilder()
    director = StoryDirector()
    director.construct(concrete_builder)
    director.basic_story()
    story = concrete_builder.story
    story.show()
    concrete_builder2 = InstagramConcreteBuilder()
```

```
    director = StoryDirector()
    director.construct(concrete_builder2)
    director.video_story()
    story = concrete_builder2.story
    story.show()


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

1.4.1 Questions

1. Soit l'objet Sandwich qui composé des variétés d'ingrédients, on veut donner des noms aux différentes configurations (choix) des sandwiches afin de faciliter la tâche de création des sandwiches :

   - Sandwich Shawarma
   - Complet viande haché
   - Frite omelette.
   - Etc.

   Implémenter la création des produits à l'aide de Builder.

2. Une voiture neuve est vendu en plusieurs options (la base, la toute, avec ou sans quelques options.

   Implémenter la création des objets voitures selon les commandes.

2 Travail à domicile واجب منزلي

لا واجب منزلي هذه المرة.

3 Références مراجع

- دورة Pattern Design باللغة العربية `https://2nees.com/courses-and-articles/design-pattern`

4 Exercices supplémentaires تمارين للتعمق

يُتبع...