



République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université AMO de Bouira

Faculté des Sciences et des Sciences Appliquées

Département d'Informatique



Compte rendu de TP 7

Méthodes et Technologies d'Implémentation

Spécialité : Génie des Systèmes Informatiques

Design Patterns

Réalisé par

— AIFAOU SARA

— FEKAR ROUMAÏSSA

2019/2020

0.1 Séance TP

0.1.1 Introduction

Le nombre d'applications développées avec des technologies orientées objets augmentant, l'idée de réutiliser des techniques pour solutionner des problèmes courants a abouti aux recensements d'un certain nombre de modèles connus sous le nom de motifs de conception (design patterns).

Ces modèles sont définis pour pouvoir être utilisés avec un maximum de langages orientés objets.

0.1.2 Méthodologie du travail

Factory method : La fabrique permet de créer un objet dont le type dépend du contexte , cet objet fait partie d'un ensemble de sous-classes. L'objet retourné par la fabrique est donc toujours du type de la classe mère mais grâce au polymorphisme les traitements exécutés sont ceux de l'instance créée.

Abstract Factory method : Le motif de conception Abstract Factory (fabrique abstraite) permet de fournir une interface unique pour instancier des objets d'une même famille sans avoir à connaître les classes à instancier.

0.1.3 Outils utilisés

import sys : Ce module fournit un accès à certaines variables utilisées et maintenues par l'interpréteur, et à des fonctions interagissant fortement avec ce dernier.

0.1.4 Tests et résultats attendus

Sans design pattern : Le résultat affiché après l'exécution (Voir la figure 1)

```
Circle.draw  
Circle.erase  
Square.draw  
Square.erase  
Circle.draw  
Circle.erase  
Square.draw  
Square.erase
```

FIGURE 1 – Résultat 1

Modifier le code pour ajouter une forme triangle et rectangle(Voir la figure 2)

```
Circle.draw  
Circle.erase  
Square.draw  
Square.erase  
Triangle.draw  
Triangle.erase  
Rectangle.draw  
Rectangle.erase
```

FIGURE 2 – Résultat 2

Simple factory method : on veut encapsuler l'opération d'instanciation des objets, en utilisant une classe ShapeFactory.(Voir la figure 3)

```
Circle.draw  
Circle.erase  
Square.draw  
Square.erase  
Triangle.draw  
Triangle.erase  
Rectangle.draw  
Rectangle.erase
```

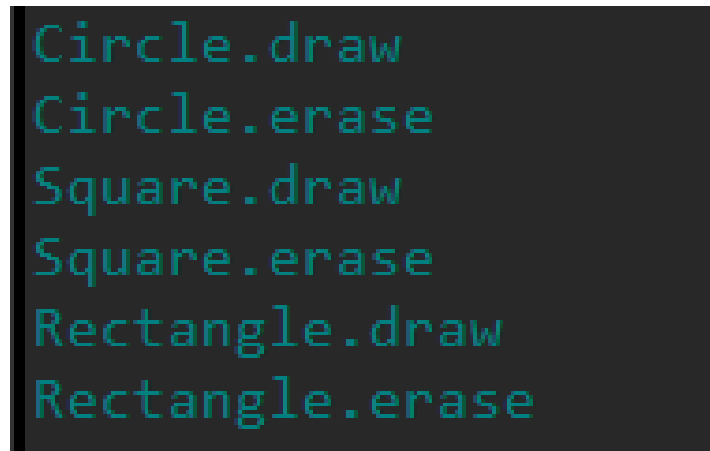
FIGURE 3 – Résultat 3

Factory method : Dans cette partie, On veut créer une Factory (ShapeFactory_SCT) spécialisée qui ne fabrique que les carrés, les cercles, et les triangles.(voir la figure 4)

```
Circle.draw  
Circle.erase  
Square.draw  
Square.erase  
Triangle.draw  
Triangle.erase
```

FIGURE 4 – Résultat 4

Une autre Factory (ShapeFactory_SCR) qui fabrique que les carrés et les cercles et les rectangles.(voir la figure 5)

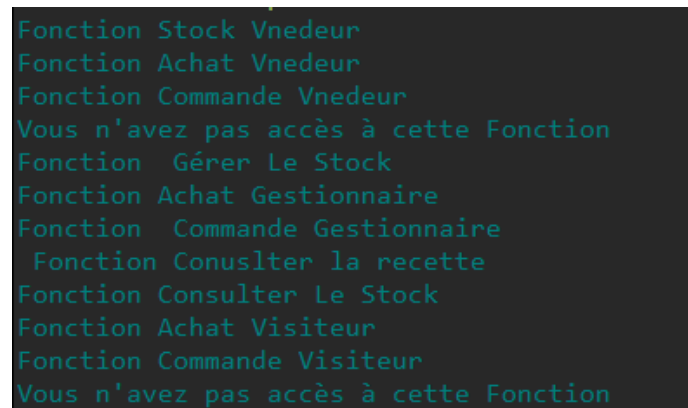


```
Circle.draw
Circle.erase
Square.draw
Square.erase
Rectangle.draw
Rectangle.erase
```

FIGURE 5 – Résultat 5

Application : On veut une application de gestion de stock, notre client exige d’avoir trois types de comptes :(voir la figure 6)

1. un compte gestionnaire, pour gérer le stock, faire des achats et des commandes, voir la recette.
2. un compte vendeur, pour permettre aux vendeurs de vendre, faire des achats et des commandes.
3. Un compte visiteur pour permettre aux clients de consulter le stock, faire des achats et des commandes.



```
Fonction Stock Vnedeur
Fonction Achat Vnedeur
Fonction Commande Vnedeur
Vous n'avez pas accès à cette Fonction
Fonction Gérer Le Stock
Fonction Achat Gestionnaire
Fonction Commande Gestionnaire
Fonction Consulter la recette
Fonction Consulter Le Stock
Fonction Achat Visiteur
Fonction Commande Visiteur
Vous n'avez pas accès à cette Fonction
```

FIGURE 6 – Résultat 6

Abstract factory : On reprends les classes de la première partie et on veut Créer des formes 2D et des formes 3D, selon le diagramme suivant.(Voir figure 7 et 8)

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

```
Circle.draw2D  
Circle.erase2D  
Square.draw2D  
Square.erase2D  
Circle.draw2D  
Circle.erase2D  
Square.draw2D  
Square.erase2D
```

FIGURE 7 – Résultat 7

```
Circle.draw3D  
Circle.erase3D  
Square.draw3D  
Square.erase3D  
Circle.draw3D  
Circle.erase3D  
Square.draw3D  
Square.erase3D
```

FIGURE 8 – Résultat 8

0.2 Travail à domicile

0.2.1 Introduction

Premierement, en utilisant le patron de conception (Factory method). on veut implémenter l'exemple de Pizza vue en cours.

1. Une Pizza peut avoir plusieurs variantes (pizza simple, pizza thon, pizza couverte, pizza marguerite,...)
2. Une pizzeria est considérée comme une usine de fabrication des produits pizza. Une pizzeria n'offre que quelques types de produits.

Exemple : Pizzeria Simple offre ; pizza thon, pizza simple.

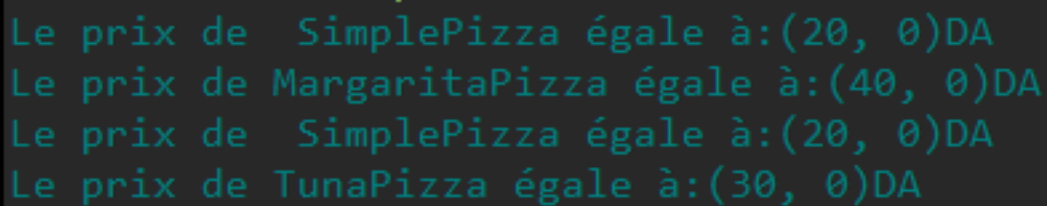
Pizzeria Luxoffre d'autres types : pizza thon, pizza marguerite.

Deusiement, En utilisant le patron «Abstract Factory». On veut implémenter L'exemple de l'interface graphique qui crée des widgets (objets graphiques) en fonction du système d'exploitation.

L'interface GUI (Graphical User Interface), propose des buttons et des zones de texte.

0.2.2 Tests et résultats attendus

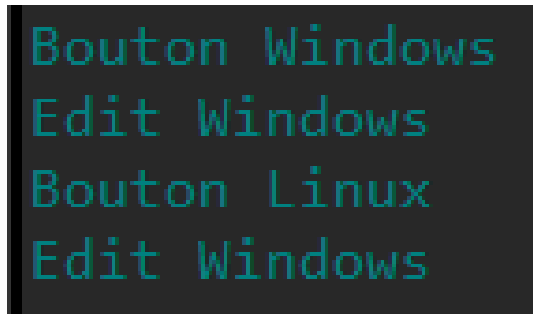
Factory method :



```
Le prix de SimplePizza égale à:(20, 0)DA
Le prix de MargaritaPizza égale à:(40, 0)DA
Le prix de SimplePizza égale à:(20, 0)DA
Le prix de TunaPizza égale à:(30, 0)DA
```

FIGURE 9 – Résultat 9

Abstract Factory method :



```
Bouton Windows  
Edit Windows  
Bouton Linux  
Edit Windows
```

A terminal window with a dark background and light blue text. It displays four lines of output: 'Bouton Windows', 'Edit Windows', 'Bouton Linux', and 'Edit Windows'.

FIGURE 10 – Résultat 10

Annexe A

Code en annexe

A.1 TP

A.1.1 Sans design pattern :

```
#!/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 Shape():
    def __init__(self):
    pass
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")
class Triangle(Shape):
    def draw(self):
        print("Triangle.draw")
    def erase(self):
        print("Triangle.erase")
class Rectangle(Shape):
    def draw(self):
        print("Rectangle.draw")
    def erase(self):
        print("Rectangle.erase")
if __name__ == "__main__":
    for type in ("Circle", "Square", "Triangle", "Rectangle"):
        if type == "Circle":
            shape = Circle()
        elif type == "Square":
            shape = Square()
        elif type == "Triangle":
            shape = Triangle()
        elif type == "Rectangle":
            shape = Rectangle()

    else:
        print("Bad_shape_creation:" + type)
    sys.exit()
    shape.draw()
    shape.erase()
```

A.1.2 Simple factory method :

```
#!/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 Shape():
    def __init__(self):
    pass
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")
class Triangle (Shape):
    def draw(self):
    print("Triangle.draw")
    def erase(self):
    print("Triangle.erase")
class Rectangle(Shape):
    def draw(self):
    print("Rectangle.draw")
    def erase(self):
    print("Rectangle.erase")
class ShapeFactory:
    @staticmethod
```

```

def createShape(type):
    if type == "Circle":
        return Circle()
    elif type == "Square":
        return Square()
    elif type == "Triangle":
        return Triangle()
    elif type == "Rectangle":
        return Rectangle()
    else:
        print ("Bad_shape_creation:" + type)
        sys.exit()
if __name__ == "__main__":
    for type in ("Circle", "Square", "Triangle", "Rectangle"):
        shape = ShapeFactory.createShape(type)
        shape.draw()
        shape.erase()

```

A.1.3 Factory method :

```

#!/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 Shape():
    def __init__(self):
        pass
    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")
class Triangle (Shape):
def draw(self):
print("Triangle.draw")
def erase(self):
print("Triangle.erase")
class Rectangle(Shape):
def draw(self):
print("Rectangle.draw")
def erase(self):
print("Rectangle.erase")
class ShapeFactory:
@staticmethod
def createShape(type):
pass
class ShapeFactory_SCT(ShapeFactory):
@staticmethod
def createShape(type):
if type == "Circle":
return Circle()
elif type == "Square":
return Square()
elif type == "Triangle":
return Triangle()
else:
print ("Bad_shape_creation:_" + type)
```

```
sys.exit()

class ShapeFactory_SCR(ShapeFactory):
    @staticmethod
    def createShape(type):
        if type == "Circle":
            return Circle()
        elif type == "Square":
            return Square()
        elif type == "Rectangle":
            return Rectangle()
        else:
            print ("Bad_shape_creation:_" + type)
            sys.exit()

if __name__ == "__main__":
    for type in ("Circle", "Square", "Triangle"):
        #shape = ShapeFactory_SCR.createShape(type)
        shape = ShapeFactory_SCT.createShape(type)
        shape.draw()
        shape.erase()
```

A.1.4 Application :

```
#!/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 Compte:
    def stock(self):
    print("Fonction_Stocke_")
    def achat(self):
    print("Fonction_Achat_")
    def commande(self):
    print("Fonction_Commande_")

class Vendeur(Compte):
    def stock(self):
    print ( "Fonction_Stock_Vnedeure_" )
    def achat(self):
    print ( "Fonction_Achat_Vnedeure_" )
    def commande(self):
    print ( "Fonction_Commande_Vnedeure_" )
    def consulter_recette(self):
    print ( "Vous_n'avez_pas_acc_s_ _cette_Fonction_" )
class Visiteur(Compte):
    def stock(self):
    print ( "Fonction_Consulter_Le_Stock_" )
    def achat(self):
    print ( "Fonction_Achat_Visiteur_" )
    def commande(self):
    print ( "Fonction_Commande_Visiteur_" )
    def consulter_recette(self):
```

```

print ( "Vous n'avez pas accès à cette Fonction")

class Gestionnaire(Compte):
def stock(self):
print ( "Fonction Gérer Le Stock" )
def achat(self):
print ( "Fonction Achat Gestionnaire" )
def commande(self):
print ( "Fonction Commande Gestionnaire" )
def consulter_recette(self):
print ( "Fonction Consulter la recette" )
class CompteFactory:
def createCompte(self):
pass
class ConcreteCompteFactory(CompteFactory):
def createCompte(self):
if (type == "Vendeur"):
return Vendeur()
elif (type == "Gestionnaire"):
return Gestionnaire()
elif (type == "Visiteur"):
return Visiteur()
if __name__ == '__main__':
for type in ("Vendeur", "Gestionnaire", "Visiteur"):
co= ConcreteCompteFactory.createCompte(type)
co.stock()
co.achat()
co.commande()
co.consulter_recette()

```


A.1.5 Abstract factory :

```
#!/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 Shape():
    def __init__(self):
    pass
class Circle(Shape):
    def draw(self) :
    print(" Circle.draw")
    def erase(self):
    print(" Circle.erase")
class Circle2D(Circle):
    def draw(self) :
    print(" Circle.draw2D")
    def erase(self):
    print(" Circle.erase2D")
class Circle3D(Circle):
    def draw(self) :
    print(" Circle.draw3D")
    def erase(self):
    print(" Circle.erase3D")
class Square(Shape):
    def draw(self) :
    print(" Square.draw")
    def erase(self):
    print(" Square.erase")
class Square2D(Square):
    def draw(self) :
```

```

print("Square.draw2D")
def erase(self):
print("Square.erase2D")
class Square3D(Square):
def draw(self) :
print("Square.draw3D")
def erase(self):
print("Square.erase3D")

class ShapeFactory:
@staticmethod
def createShape(type):
pass
class ShapeFactory2D(ShapeFactory):
@staticmethod
def createShape(type):
if type == "Circle":
return Circle2D()
elif type == "Square":
return Square2D()
else: return None
class ShapeFactory3D(ShapeFactory):
@staticmethod
def createShape(type):
if type == "Circle":
return Circle3D()
elif type == "Square":
return Square3D()
else: return None
if __name__ == '__main__':

for type in ("Circle", "Square", "Circle", "Square"):

```

```
#shape = ShapeFactory2D.createShape(type)
shape = ShapeFactory3D.createShape(type)
shape.draw()
shape.erase()
```

A.2 Travail à domicile

A.2.1 Factory method :

```
#!/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 Pizza:
    def __init__(self):
    pass
    class PizzaSimple(Pizza):
        price=20,00
        def get_price(self):
            p=self.price
            print("Le_prix_de_SimplePizza_gale : "+ str(p) +"DA")
    class PizzaThon(Pizza):
        price=30,00
        def get_price(self):
            p=self.price
            print("Le_prix_de_TunaPizza_gale : "+ str(p) +"DA")
    class PizzaMargarita(Pizza):
        price=40,00
        def get_price(self):
            p=self.price
```

```

print("Le_prix_de_MargaritaPizza_gale : "+ str(p) +"DA")

class PizzaFactory:
    def createPizza(self):
        pass

class PizzeriaLux (PizzaFactory):
    def createPizza(self):
        if type == "PizzaThon":
            return PizzaThon()
        elif type == "PizzaMargarita":
            return PizzaMargarita()
        else:
            print ( "ce_type_n'existe_pas" + type )

class PizzeriaHouma (PizzaFactory):
    def createPizza(self):
        if type == "PizzaSimple":
            return PizzaSimple()
        elif type == "PizzaThon":
            return PizzaThon()
        else:
            print ( "ce_type_n'existe_pas" + type )

if __name__ == '__main__':
    for type in ("PizzaThon","PizzaMargarita"):
        pizza=PizzeriaLux.createPizza(type)
        pizza.get_price()
    for type in ("PizzaSimple","PizzaThon"):
        pizza=PizzeriaHouma.createPizza(type)
        pizza.get_price()

```

A.2.2 Abstract Factory method :

```
#!/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
import platform
class Widget:
    def __init__(self):
    pass
class Button(Widget):
    def print(self):
    pass
class WinButton(Button):
    def print(self):
    print("Bouton_Windows" )
class LinuxButton(Button):
    def print(self):
    print("Bouton_Linux_")
class Edit(Widget):
    def print(self):
    pass
class WinEdit(Edit):
    def print(self):
    print("Edit_Windows")
class LinuxEdit(Edit):
    def print(self):
    print("Edit_Windows")
class GuiFactory:
    def createGui(self):
    pass
```

```
class WinGuiFactory (GuiFactory):
def createGui(self):
if type == "WinButton":
return WinButton()
elif type == "WinEdit":
return WinEdit()

class LinuxGuiFactory (GuiFactory):
def createGui(self):
if type == "LinuxButton":
return LinuxButton()
elif type == "LinuxEdit":
return LinuxEdit()

if __name__ == '__main__':
systeme=platform.system()
print("Le_systeme_est:"+systeme)
if systeme=="windows":
for type in ("WinButton","WinEdit"):
widget=WinGuiFactory.createGui(type)
widget.print()
elif systeme=="Linux":
for type in ("LinuxButton","LinuxEdit"):
widget=LinuxGuiFactory.createGui(type)
widget.print()
```