Série TP n°7 2022-2023	Module	MTI Méthd. et tech d'implement.
	Filière	Master GSI 1 ère Année
Chapitre 3: Design patterns	ملاحظة :	

- **Objectifs** أهداف: Design patterns de structure
- **Données** بيانات:
- **Outils** أدوات : Python

1 Partie TP

1.1 Adapter

Exécuter le programme suivant :

```
class Person:
    def __init__(self, nom, prenom):
        self.nom = nom
        self.prenom = prenom

class personne:
    def __init__ (self, nomcomplet):
        self.nom = nomcomplet

class adapter(personne):
    def __init__(self, person):
        self.nom    = person.nom +"-"+ person.prenom

def main():
    # personne
    personOne = Person("Akli", "Mohand")
    per = adapter(personOne)
    print(per.nom)
    return 0

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

Questions :

1. Analyser le pattern et citer ces avantages.
2. La date est stocké sur la machine en format YYYY/MM/DD, on veut créer des adaptateurs qui permet d'afficher les dates dans plusieurs styles (MM/DD/YYYY), (DD/MM/YYYY).

1.2 Façade

```
class Facade:
    """
    Know which subsystem classes are responsible for a request.
    Delegate client requests to appropriate subsystem objects.
    """

    def __init__(self):
        self._subsystem_1 = Subsystem1()
        self._subsystem_2 = Subsystem2()
        self._subsystem_3 = Subsystem3()

    def operation_retait_ccp(self):
        self._subsystem_1.operation1()
    def operation_virment_ccp(self):
        self._subsystem_1.operation2()
    def operation_fact_eau(self):
        self._subsystem_2.operation1()
    def operation_fact_elec_gaz(self):
        self._subsystem_3.operation1()


class Subsystem1:
    """
    Implement subsystem functionality.
    Handle work assigned by the Facade object.
    Have no knowledge of the facade; that is, they keep no references to
    it.
    """

    def operation1(self):
        print("operation retrait ccp")

    def operation2(self):
        print("operation vierment ccp")


class Subsystem2:
    """
    Implement subsystem functionality.
    Handle work assigned by the Facade object.
    Have no knowledge of the facade; that is, they keep no references to
    it.
    """

    def operation1(self):
        print("operation payment facture  de l'eau ADE")
class Subsystem3:
    """
    Implement subsystem functionality.
    Handle work assigned by the Facade object.
    Have no knowledge of the facade; that is, they keep no references to
    it.
    """

    def operation1(self):
```

```
        print("operation payment facture  de Eletcricité/gaz Sonelgaz")


def main():
    facade = Facade()
    facade.operation_retait_ccp()
    facade.operation_virment_ccp()
    facade.operation_fact_eau()
    facade.operation_fact_elec_gaz()


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

Questions

1. Il existe plusieurs ressources d'informations pour les étudiants en informatique ;
   - Site de département
   - Plateforme elearning-Info
   - Plateforme elearning de l'université
   - Page facebook de la Faculté
   - les groupes des étudiants

   Proposer un façade pour regrouper les affichages en un seul endroit.
2. Algérie Poste utilise deux stratégies :
   - guichet unique pour toutes les opérations
   - guichet spécialisé pour chaque opération
   - Critiquer les deux stratégies

1.3 Proxy

```
class RealSubject:
    """
    The RealSubject contains some core business logic. Usually, RealSubjects are
    capable of doing some useful work which may also be very slow or sensitive -
    e.g. correcting input data. A Proxy can solve these issues without any
    changes to the RealSubject's code.
    """

    def request(self):
        print("RealSubject: Handling request.")


class Proxy(RealSubject):
    """
    The Proxy has an interface identical to the RealSubject.
    """

    def __init__(self, RealSubject):
        self._real_subject = real_subject

    def request(self):
        """
        The most common applications of the Proxy pattern are lazy loading,
        caching, controlling the access, logging, etc. A Proxy can perform one
        of these things and then, depending on the result, pass the execution to
        the same method in a linked RealSubject object.
        """

        if self.check_access():
            self._real_subject.request()
            self.log_access()

    def check_access(self):
        print("Proxy: Checking access prior to firing a real request.")
        return True

    def log_access(self):
        print("Proxy: Logging the time of request.", end="")


def client_code(subject: RealSubject):
    """
    The client code is supposed to work with all objects (both subjects and
    proxies) via the Subject interface in order to support both real subjects
    and proxies. In real life, however, clients mostly work with their real
    subjects directly. In this case, to implement the pattern more easily, you
    can extend your proxy from the real subject's class.
    """

    # ...

    subject.request()

    # ...
```

```
if __name__ == "__main__":
    print("Client: Executing the client code with a real subject:")
    real_subject = RealSubject()
    client_code(real_subject)

    print("")

    print("Client: Executing the same client code with a proxy:")
    proxy = Proxy(real_subject)
    client_code(proxy)
```

Questions

1. Soit un service d'accès à une salle de sport ou un stade.

   Réaliser le proxy qui permet de vérifier le passe sanitaire (covid19) avant d'entrer en salle.

2. Pour déposer le dossier de visa, les ambassades font des contrats avec des sociétés spécialisées .

   Réaliser le schéma en proxy pour l'opération de dépôt des demandes de visas.

1.4 Design patterns de comportement : Strategy

```python
#Define a family of algorithms, encapsulate each one, and make them
#interchangeable. Strategy lets the algorithm vary independently from
#clients that use it.

class Context:
    """
    Define the interface of interest to clients.
    Maintain a reference to a Strategy object.
    """
    def __init__(self, strategy):
        self._strategy = strategy

    def context_interface(self):
        self._strategy.algorithm_interface()

class Strategy():
    """
    Declare an interface common to all supported algorithms. Context
    uses this interface to call the algorithm defined by a
    ConcreteStrategy.
    """
    def algorithm_interface(self):
        pass

class ConcreteStrategyA(Strategy):
    """
    Implement the algorithm using the Strategy interface.
    """
    def algorithm_interface(self):
        pass

class ConcreteStrategyB(Strategy):
    """
    Implement the algorithm using the Strategy interface.
    """

    def algorithm_interface(self):
        pass

def main():
    concrete_strategy_a = ConcreteStrategyA()
    context = Context(concrete_strategy_a)
    context.context_interface()

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

Questions

1. On veut implémenter le passage entre les années d'études en licence et en master (la moyenne et les crédits nécessaires, ainsi que les unités requises.
2. On veut implémenter une procédure de crédit bancaire avec au sans intérêt.

1.5 Design patterns de comportement :Observer

```
"""
Define a one-to-many dependency between objects so that when one object
changes state, all its dependents are notified and updatedautomatically.
"""
class Subject:
    """
    Know its observers. Any number of Observer objects may observe a
    subject.
    Send a notification to its observers when its state changes.
    """
    def __init__(self):
        self._observers = set()
        self._subject_state = None

    def attach(self, observer):
        observer._subject = self
        self._observers.add(observer)

    def detach(self, observer):
        observer._subject = None
        self._observers.discard(observer)

    def _notify(self):
        for observer in self._observers:
            observer.update(self._subject_state)

    def subject_state(self, arg):
        self._subject_state = arg
        self._notify()


class Observer():
    """
    Define an updating interface for objects that should be notified of
    changes in a subject.
    """

    def __init__(self):
        self._subject = None
        self._observer_state = None

    def update(self, arg):
        pass
    def show(self,):
        print(self._observer_state)


class ConcreteObserver(Observer):
    """
    Implement the Observer updating interface to keep its state
    consistent with the subject's.
    Store state that should stay consistent with the subject's.
    """

    def update(self, arg):
```

```
        self._observer_state = arg
        # ...


def main():
    subject = Subject()
    ahmed_observer = ConcreteObserver()
    salem_observer = ConcreteObserver()
    subject.attach(ahmed_observer)
    subject.attach(salem_observer)

    subject.subject_state(123)

    ahmed_observer.show()
    salem_observer.show()

    subject.subject_state(99)

    ahmed_observer.show()
    salem_observer.show()



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

Questions

1. On veut implémenter une classe de notification à partir de site de département, où l'étudiant peut s'inscrire pour recevoir les notifications des nouvelles postes.
2. On veut implémenter un calendrier ou les utilisateurs peuvent s'inscrire pour recevoir les rappels.

2 Travail à domicile واجب منزلي

Choisir 3 designs patterns (un par catégorie), puis implémenter les questions de chaque pattern.
اختر ثلاثة أنماط تصميم (واحد من كل صنف) ثم أنجز أسئلة كل نمط.

2.1 Questions supplémentaires (facultatif) أسئلة إضافية (اختيارية)

Pour la gestion d'une agence touristique, on veut développer :

- Une façade pour les services sous-traités : transport, hébergement et restauration.
- Plusieurs stratégies pour chaque service, afin de satisfaire les différents budgets des clients.
- Des Adaptateurs pour le payement en dinars des services à l'étranger.
- Un contrôle d'accès aux ressources afin d'éviter les chevauchements des réservations (utiliser le singleton patterns).
- Possibilité d'annuler les réservations (utiliser le memento pattern)

Questions :

- Tracer les diagrammes des classes.
- Expliquer l'utilisation des patterns.
- Simuler l'implémentation des classes.

3 Références مراجع

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

4 Exercices supplémentaires تمارين للتعمق

يُتبع...