Genie Logiciel

TP5 Vol - Reservation - Association bidirectionnelle

JEROME Remy WEINKOPF Nicolas - 2 décembre 2015

Introduction

Objectif : Implémentez le modèle défini en TD.

Le TP étant réalisé dans la continuité du TP sur Git, nous avons utilisé git pour mener a bien ce projet:

https://github.com/remyjerome/gl

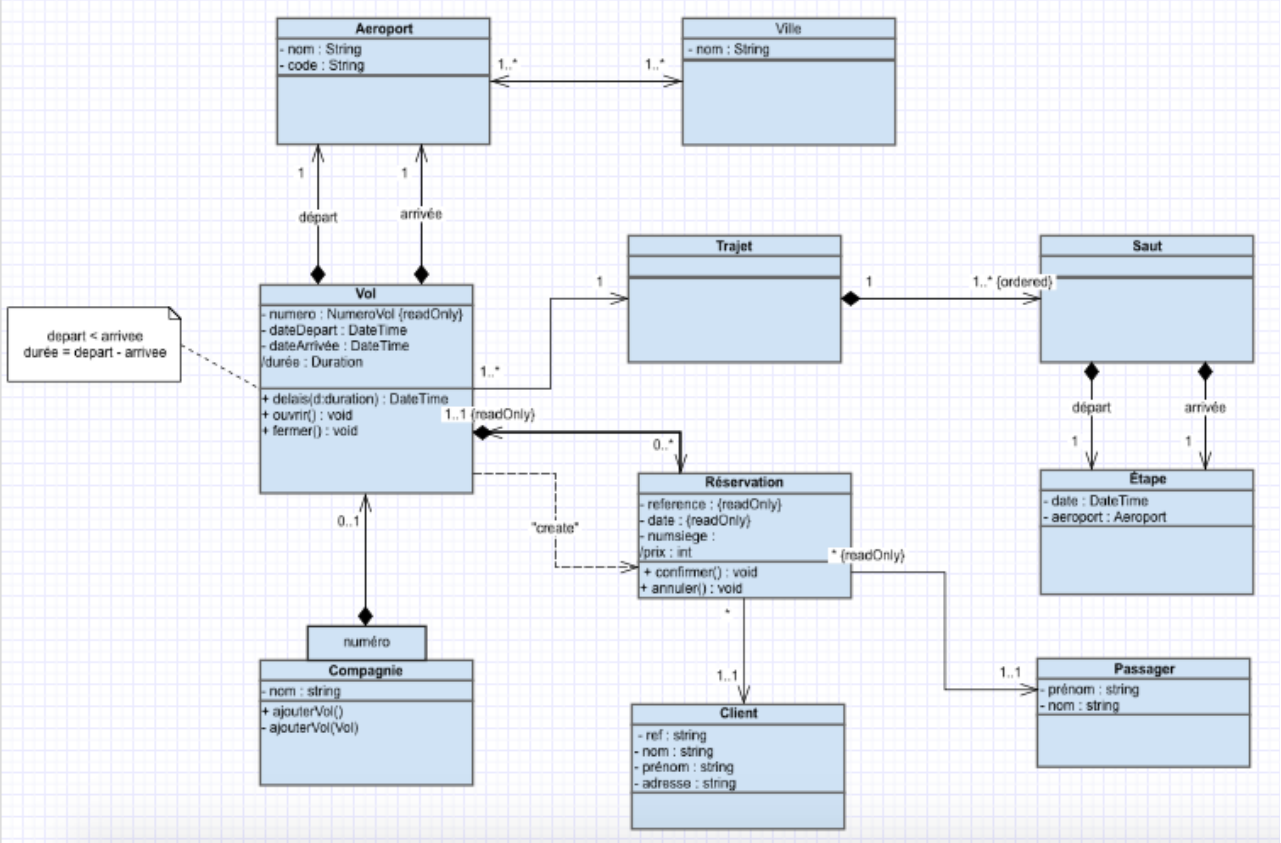
Pour celà, nous avons commencé par implémenter la première version du package « Vol », sans la métaclasse pour les vols.

Pour ensuite implémenter ensuite le package de réservation.

Puis nous avons implémenter la double navigabilité entre deux classes.

Et en fin nous avons implémenté les métavols en passant par une classe MetaVol.

Voici le diagramme UML implémentés :



Package Vol

```
import time
import datetime
import re
import os
import pickle

class NumeroVol: #Definiton class Numero Vol
     """Class permettant d'attribuer un numero de vol unique a
chaque vol"""
     def __init__(self, numero):
          try:
               self._numero = numero
               if re.match(r"^[a-zA-Z0-9_]{10}$", numero) == None:
#Expression reguliere de la forme 10 carractere aplhanumerique
obligatoire
                    raise ValueError("le numero saisi est
invalide")
          except ValueError:
               print("Erreur : La valeur saisie est invalide(le
numero doit contenir dix carracteres de type apha numerique)")
               raise
     def _get_numero(self):
          return self._numero
     def __call__(self):
          return(self)
     def __eq__(self, other):
          return self.__dict__ == other.__dict__
     def __str__(self):
          return "Numero {}".format(self._numero)
     numero=property(_get_numero)
class Vol(object) : #Definiton classe Vol
     """La class vol permet de creer un vol avec un numero de vol
de type NumeroVol"""
     def __init__(self, aeroportDepart, aeroportArrivee,
dateDepart, dateArrivee, numeroVol) :
          self._numeroVol = numeroVol
          self._aeroportDepart = aeroportDepart
          self._aeroportArrivee = aeroportArrivee
          self._etatVol = True #Le vol est de base ouvert aux
reservation
          self._trajet = None
          try:
               self._dateDepart = dateDepart
               self._dateArrivee = dateArrivee
               if(dateDepart > dateArrivee):
                    raise ValueError("les dates saisies sont
invalides")
          except ValueError:
```

```
                print("Erreur : Les Dates saisies sont invalides(la
date de depart doit-etre inferieure a la date d'arrivee)")
                raise
            self._duree = dateArrivee - dateDepart

    def ouvrirReservation(self):
            self._etatVol = True
            print ("Le vol %s est maintenant ouvert au
reservations"%(self._numeroVol.numero))
    def fermerReservation(self):
            self._etatVol = False
            print ("Le vol %s est maintenant ferme au reservations"%
(self._numeroVol.numero))
    def _get_numeroVol(self):
            return self._numeroVol
    def _get_etatVol(self):
            if self._etatVol:
                print ("Le vol est ouvert aux reservations.")
            else:
                print("Le vol est ferme aux reservations")
    def __call__(self):
            print(self._numeroVol)
    def __str__(self):
            return str(self.__dict__)
    numeroVol=property(_get_numeroVol)
    etatVol=property(_get_etatVol)
class CompagnieAerienne(object) : #Definiton classe
CompagnieAerienne
    """La classe companie peut creer ces propres vols et ouvrir/
fermer une reservation"""
    def __init__(self, nom) :
            self._nom = nom
            self._listeVols = []
            print("Companie %s cree."%(self._nom))
    def ajouterVol(self,aeroportDepart, aeroportArrivee,
dateDepart, dateArrivee,numeroVol) :
            #Creation d'un numero de vol
            num = NumeroVol(numeroVol)
            try:
                for vol in self._listeVols:
                    if(vol.numeroVol == num):
                        raise ValueError("le numero de vol %s
existe deja" %(num.numero))
            except ValueError:
                print("Erreur : Le numero de vol est invalide(ce
numero de vol existe deja)")
                raise
            vol = Vol(aeroportDepart, aeroportArrivee, dateDepart,
dateArrivee,num())
            self._listeVols.append(vol)
```

```
            #Enregistrement des vols dans un fichier qui ce nomme
comme le nom de la companie
            with open(self._nom, 'wb') as fichier:
                  mon_pickler = pickle.Pickler(fichier)
                  mon_pickler.dump(vol)

            print ("Vol numero :  %s ajoute a la companie : %s" %
(num.numero, self._nom))
      def chercherVol(self, numeroVol):#Permet de chercher un vol
dans la liste de la companie et de savoir si il existe ou non
            num = NumeroVol(numeroVol)
            try:
                  for vol in self._listeVols:
                        if(vol.numeroVol == num):
                              return vol
                  raise ValueError("le numero de vol %s n'existe pas
dans la companie %s" %(num.numero,self._nom))
            except ValueError:
                  print("Erreur : Le numero de vol est invalide(ce
numero de vol n'existe pas)")
                  raise
      def ouvrirReservation(self,numeroVol) :
            self.chercherVol(numeroVol).ouvrirReservation()
      def fermerReservation(self,numeroVol) :
            self.chercherVol(numeroVol).fermerReservation()
      def __call__(self):
            print(self._nom)
      def __str__(self):
            return str(self.__dict__)

class Aeroport(object) : #Definiton classe Aeroport
      def __init__(self, nomAeroport, idAeroport, villes) :
            self._nomAeroport = nomAeroport
            self._idAeroport = idAeroport
            self._villes = villes #Liste villes
            print ("Aeroport %s numero %d cree. Il dessert %s, %s,
%s."%(self._nomAeroport, self._idAeroport,  self._villes[0],
self._villes[1], self._villes[2]))
      def __call__(self):
            print(self._nomAeroport + self._idAeroport)
      def __str__(self):
            return "Nom {}, ID {}".format(self._nomAeroport,
self._idAeroport)
class Time(object) : #Definition de notre objet time qui herite de
l'objet time de python
      def __init__(self,year=0, month=1, day=1, hour=0, minute=0) :
            date = datetime.datetime(year, month, day, hour, minute)
            self._datetime = date
      @property
      def date(self):
```

```
            return (self._date)
      def __gt__(self, other):
            return self.__dict__ > other.__dict__
      def __call__(self):
            return(self._datetime)
      def __str__(self):
            return str(self._datetime.strftime("%a, %d %b %Y %H:%M:
%S +0000"))
class Trajet(object):
      """Un trajet est une liste de saut et un saut est une etape
d'arrive et une etape de depart"""
      def __init__(self, listeSauts) :
            self._listeSauts = listeSauts
      def __call__(self):
            return self
      def __str__(self):
            print("Tajet de vol :")
            i=1
            for saut in self._listeSauts:
                  print(" Saut %d:"%(i))
                  print saut
                  i+=1
            return ""
class Saut(object):
      def __init__(self, etapeDepart, etapeArrivee) :
            try:
                  self._etapeDepart = etapeDepart
                  self._etapeArrivee = etapeArrivee
                  if(self._etapeDepart.date >
self._etapeArrivee.date):
                        raise ValueError("La date de depart est
superieure a la date d'arrivee.")
            except ValueError:
                  print("Erreur : Les date saisie sont
invalides(dateDepart < dateArrivee)")
                  raise
      def __call__(self):
            return self
      def __str__(self):
            return "  Etape depart: {} \n  Etape arrivee:
{}".format(self._etapeDepart, self._etapeArrivee)
class Etape(object):
      """Une etape est une date et un aeroport"""
      def __init__(self,date,aeroport) :
            self._date = date
            self._aeroport = aeroport
      @property
      def date(self):
            return (self._date)
      def __call__(self):
```

```
            return self
        def __str__(self):
            return "\n   Aeroport: {} \n   Date:
{}".format(self._aeroport, self._date)
if __name__ == '__main__':
        a1 = Aeroport("Clermont", 001, [("Lyon"),("Toulouse"),
("Lille")])
        a2 = Aeroport("Lyon", 002, [("Nice"),("Paris"),("Agde")])
        a3 = Aeroport("Nantes", 003,[("Anger"),("Marseille"),
("Auxerre")])
        t1=Time(2015, 12,10,12,0)
        t2=Time(2015, 12,10,15,30)
        c1 = CompagnieAerienne("air france")
        c1.ajouterVol(a1,a2,t1(),t2(),"1234567890")
        c1.ajouterVol(a2,a3,t1(),t2(),"1234567892")
        c2 = CompagnieAerienne("easy jet")
        c1.chercherVol("1234567890").etatVol
        c1.ouvrirReservation("1234567890")
        c1.chercherVol("1234567890").etatVol
        c1.fermerReservation("1234567890")
        c1.chercherVol("1234567890").etatVol

        #Creation etapes
        t3=Time(2015, 12,10,16,00)
        t4=Time(2015, 12,10,17,45)
        e1 = Etape(t1,a1)
        e2 = Etape(t2,a2)
        e3 = Etape(t3,a2)
        e4 = Etape(t4,a1)
        s1 = Saut(e1, e2)
        s2 = Saut(e3, e4)
        t1 = []
        t1.append(s1)
        t1.append(s2)
        trajet = Trajet(t1)
        print trajet
```

Package Reservation

```
from Vol import *
import re
import datetime
import pickle
class Reference(object):
    """La class Permet d'attibuer un numero des ref au client"""
    def __init__(self, reference):
        try:
            self._reference = reference
            if re.match(r"^[a-zA-Z0-9_]{3}$", reference) == None:
#REGEX de 3 carractere aplhanumerique
                raise ValueError("la reference saisi est
invalide")
        except ValueError:
            print("Erreur : La valeur saisie est invalide(la
reference doit contenir 3 carracteres de type apha numerique)")
            raise
    @property
    def reference(self):
        return self._reference
    def __call__(self):
        return(self)
    def __eq__(self, other):
        return self.__dict__ == other.__dict__
    def __str__(self):
        return "{}".format(self._reference)
class Client(object):
    """Le client contient une liste de reservation pour des
passagers"""
    def __init__(self,nom, prenom, adresse, ref) :
        self._nom = nom
        self._prenom = prenom
        self._adresse = adresse
        self._ref = Reference(ref)
        self._listeReservations = []
    def reserverVol(self, numero, passager, numeroVol, companie):
        date = datetime.datetime.now()
        reservation = Reservation(date, numero, passager,
numeroVol, companie)
        self._listeReservations.append(reservation)
        print ("Reservation cree pour le client :\n %s" %(self))
    def listeVols(self, companie):
        """Methode qui permet de recuperer la liste de vol"""
        with open(companie, 'rb') as fichier:
            mon_depickler = pickle.Unpickler(fichier)
            vols_recupere = mon_depickler.load()
            print vols_recupere
            return vols_recupere
```

```
    def __call__(self):
        return self
    def __str__(self):
        return "Nom {}\n Prenom {}\n Adresse {}\n Reference
{}".format(self._nom, self._prenom, self._adresse, self._ref)
class Reservation(object):
    """La class reservation contient un client un passager et un
numero"""
    def __init__(self, date, numero, passager, numeroVol,
companie) :
        self._date = date
        self._numero = numero
        self._passager = passager
        self._etat = None
        self._companie = companie
    def annuler(self):
        self._etat = False
        print ("Reservation numero %s annulee."%(self._numero))
    def confirmer(self):
        self._etat = True
        print ("Reservation numero %s conformee."%(self._numero))
    def __call__(self):
        return self
    def __str__(self):
        return "Reservation numero {} le {}\n Client :\n Passager
{}".format(self._numero, self._date, self._client, self._passager)
class Passager(object):
    def __init__(self, nom, prenom) :
        self._nom = nom
        self._prenom = prenom
        print ("Passager : %s %s cree"%(self._nom, self._prenom))
    def __call__(self):
        return self
    def __str__(self):
        return "Nom {} Prenom {}".format(self._nom, self._prenom)
if __name__ == '__main__':
    c1 = Client("Remy", "Jerome", "rue 25 pasteur ", "111")
    p1 = Passager("Remy","Jerome")
    c1.reserverVol("123",p1,"0123456789","air france")
```

Double navigabilité

```
class A(object):
    def __init__(self):
        self._B = None
    def addB(self, B):
        self._B = B
        self._B.setA(self)
    @property
    def B(self):
        return self._B
    @B.setter
    def setB(self, B):
        self._B = B
    def __call__(self):
        return self
    def __str__(self):
        return "Objet A : B {}".format(self._B)

class B(object):
    def __init__(self):
        self._A = None
    def addA(self, A):
        self._A = A
        self._A.setB(self)
    @property
    def A(self):
        return self._A
    @A.setter
    def setA(self, A):
        self._A = A
    def __call__(self):
        return self
    def __str__(self):
        return "Objet B : A {}".format(self._A)
if __name__ == '__main__':
    a = A()
    b = B()
    a.addB(b)
    b.addA(a)
    print a
    print b
```

MetaVol

```
from Vol import *
def MetaVol(nom, param) :
    class Vol2(object) :
        __name__ = nom
        attr_de_classe = param

        def __init__(self, aeroportDepart, aeroportArrivee,
dateDepart, dateArrivee, numeroVol) :
            self._numeroVol = numeroVol
            self._aeroportDepart = aeroportDepart
            self._aeroportArrivee = aeroportArrivee
            self._etatVol = True #Le vol est de base ouvert aux
reservation
            try:
                self._dateDepart = dateDepart
                self._dateArrivee = dateArrivee
                if(dateDepart > dateArrivee):
                    raise ValueError("les dates saisies sont
invalides")
            except ValueError:
                print("Erreur : Les Dates saisies sont
invalides(la date de depart doit-etre inferieure a la date
d'arrivee)")
                raise
            self._duree = dateArrivee - dateDepart

        def ouvrirReservation(self):
            self._etatVol = True
            print ("Le vol %s est maintenant ouvert au
reservations"%(self._numeroVol.numero))
        def fermerReservation(self):
            self._etatVol = False
            print ("Le vol %s est maintenant ferme au
reservations"%(self._numeroVol.numero))
        def _get_numeroVol(self):
            return self._numeroVol
        def _get_etatVol(self):
            if self._etatVol:
                print ("Le vol est ouvert aux reservations.")
            else:
                print("Le vol est ferme aux reservations")
        def __call__(self):
            print(self._numeroVol)
        def __str__(self):
            return "Vol numero {} en provenance de {} et a
destination de {} depart prevu a {} et arrivee prevu a
{}".format(self._numeroVol, self._aeroportDepart,
self._aeroportArrivee, self._dateDepart, self._dateArrivee)
```

```
        numeroVol=property(_get_numeroVol)
        etatVol=property(_get_etatVol)

    return Vol

if __name__ == '__main__':
    #Creation des objet Aeroport et Time
    a1 = Aeroport("Clermont", 001, [("Lyon"),("Toulouse"),
("Lille")])
    a2 = Aeroport("Lyon", 002, [("Nice"),("Paris"),("Agde")])
    a3 = Aeroport("Nantes", 003,[("Anger"),("Marseille"),
("Auxerre")])
    t1=Time(2015, 12,10,12,0)
    t2=Time(2015, 12,10,15,30)
    #Creation d'une meta classVol
    Vol2 = MetaVol('VolGenerique', 'test')
    #Exemple 1 d'instanciation de Vol
    instance = Vol2(a1,a2,t1(),t2(),"1234567890")
    print instance
    #Exemple 2 d'instanciation de Vol
    instance2 = Vol2(a1,a2,t1(),t2(),"1234567893")
    print instance2
```