

MI. Lic 2 Acad/Section A/Groupe 2

**RAPPORT SEMESTRIEL N°2 : POO**

19-05-2018

PROJET SUR LA GESTION DES APPELS TELEPHONIQUES DANS UN OPERATEUR TELEPHONIQUE

Réalisé par :

AKLI Yassamine && MEDDAS Nesrine

Encadré par :

Madame S.BOUKHADOUMA-chargé de cours et de td POO

Madame OUAZAR-chargé de tp POO

USTHB

Facultée d’él éctronique et d’informatique

**Département informatique**

Contenu

[Introduction 1](#_Toc514579077)

[Description des classes avec leurs shéma 2](#_Toc514579078)

[Présntation de l’application 13](#_Toc514579079)

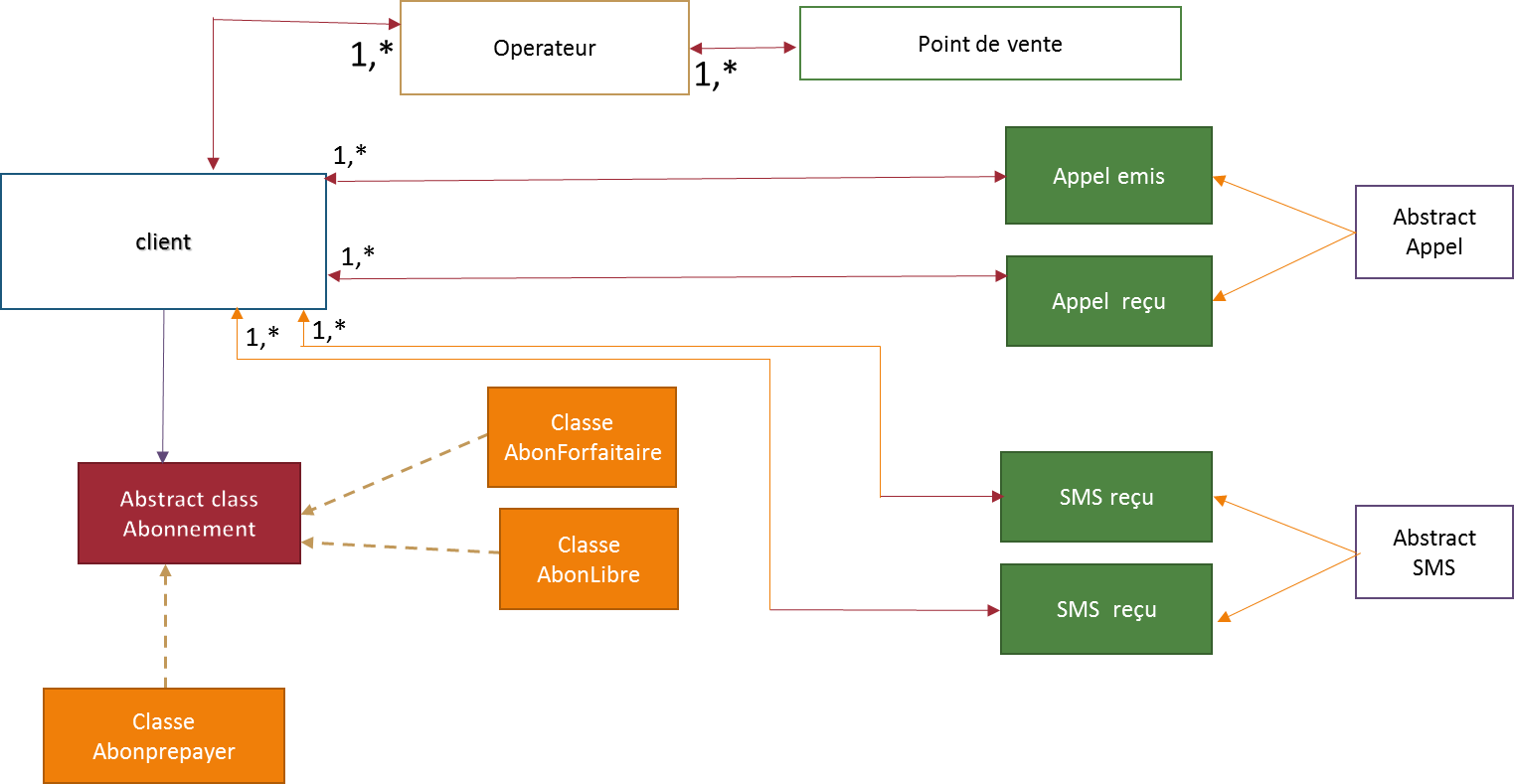
[Conclusion 19](#_Toc514579080)

# Introduction

Ce projet consiste à réaliser une simulation d’une application réel qui gère les clients et appels /sms d’un opérateur téléphonique ce petit logiciel est établit en l’usage de l’héritage, exceptions,

# Description des classes avec leurs shéma

## Shéma conceptuel des classes implementee :



Lgende : : Appel emis hérite de Appel  : Client a une variable de classe de type Abonnement , : Operateur a plusieur variables de classe de type Client/point de vente

Figure Modélisation des classe avec leurs associations

## Description des classes

* Client :

Abonnement :Type; /\*attribut\*/String:nom,prénom;AdresseMail:mail; Adresse:Adress;Set<SMSEmis>SMSsortant; Set<SMSReçu>SMSentrant; Set<AppelEmis>APPELsortant; //de type HashSet Set<AppelReçu>APPELentrant; String NumTel;String numéroContrat; Date dateContrat;

//constructeur avec parametre:

Client(String nom, String prénom, String numTelephone,Date d,AdresseMail mail,Adresse adr, Abonnement A)

throws NegativeException, TelephoneExceptions, DateException, AdresseException

//constructeur sans parametre:

Client() );

//methode saisir et affichage et to String:

Saisir()(void)-Client;

Affichage()(void)-Client;

toString()(String)-Client;

AfficherAppel()(void)-Client

/\*-----appel: dans ces méthodes on a opter a vérifier les dates; l’appel doit être lancer après la date de contrat et la date d’abonnement---si la date d’appel vient pendant ou après la date fin d’abonnement de blocage alors le client sera bloquée avec stBloqué

si client est bloqué=>envoyé null+lancer une exception « client bloqué impossible d’effectuer cet appel » -----------meme principe pour AppelSortant et SMSEntrant et SMSSortant\*/

/\*-----la vérification du solde s’implémente dans la méthode AjouterAppelSortrant et AjouterSMSSortant via l’abonnement=>ces methodes sont appelé pour crée l’appel ou sms sortant et récupéré le retour dans un objet appel crée dans ces methode qui suivent (c’est lui-même qui sera AjouterAppelEntrant(String num, Date datAppel, Heure

HeureAppel, int durée):(AppelEntrant)-Client;

AjouterAppelSortant(String num, Date datAppel, Heure HeureAppel, int durée):(AppelSortant)-Client

AjouterSMSEntrant(String num, Date datAppel, Heure HeureSMS,String text):(SMSEntrant)-Client

AjouterSMSSortant(String num, Date datAppel, Heure HeureSMS, Strnig text):(SMSSortant)-Client

//------------ses méthodes sont appelé dans les methodes:

//-----------: AjouterAppelsEmis, AjouterAppelRecu,

/\*AjouterSMSEmis, AjouterSMSRecu; permettent d’ajouter un appel ou sms dans la collection correspondante ; ses méthodes verifie d’abord la validitée du numéro donnée si la date de blocage est atteinte alors on envoie trois sms entrant à des dates différente , puis on bloque le client , sinon on ajoute exemple :(this,A,add(this,AjouterApellEntrant(,,,,,);\*//\*des exceptions peuvent être levé au cas de blocage, solde insuffisant ou date invalide elle seront catché dans cette methode\*/

* Operateur :

**String nom;**

**//ensemble de point de vente:**

**ArrayList <PointDeVente> ptrVente=new ArrayList <PointDeVente>();**

**//pourcentage de couverture par wilaya**

**long pourcentage[] = new long[49];**

**//ensembles de clients abonnées:**

**ArrayList <Client> Abonnées=new ArrayList <Client>();**

**Operateur()-Operateur;//constructeru**

**Operateur(String nomoperatur, ArrayList<PointDeVente> ptrVente, ArrayList<Client> abonnées)-Operateur**

**Operateur(String nom)-Operateur;**

**toString()(String)-Operateur; /\*renvoie la description de l’objet operateur avec ses abonnées et wilaya avec leurs pourcentage et points de ventes\*/**

**//---Afichage:**

**AfficherPointDeVente(()void)-Operateur//afficher les //points de ventes**

**//pour afficher les client triée par type d’abonnement: //ajouter les abonnées dans setAbon puis on affiche**

**AfficherClientParTypeAbonnement()(void)-Oprateur;**

**//si client bloqué on affiche :**

**AfficherClientBloqué()(void)-Operateur;**

**//Afficher les clients non bloquée :**

**AfficherClientNonBlqué**

**//afficher les wilaya avec leur pourcentage:**

**AfficherWilayaPourcentage()(void)-Operateur;**

pourcentageModifier(int numWilaya,long nouvPouccentag)(void);/\*la verification de la validitée du numéro de la wilaya est en dehors de cette méthode \*/

SupprimerClient(Client c)(void);/\*supprimer un client on suppose qu’il existe\_ le contrôle de l’existance est effectué en dehors de ces méthodes\*/

SupprimerPtrVente(PointDeVente ptrVente)(void);

/\*on suppose que les objets à ajouté n’existe pas\*/

AjouterPtrVente(PointDeVente p)(void);

//-----les setters et getters

//-------Methode de recherche:

/\*rechercher un numero donné et retourner l’objet si il existe null sinon\*/

RecherchernNumeroDonnéeClient(String numéro)(Client)-Operateur;

/\*rechercher un numero donné si on le trouve on retourne son indice pour pouvoir le modifier plutart et effectué les traitements qu’on veut\*/

getMemeNuméroEmplacementPtrVente(String num)(int)-Operateur;

//rechercher un numero donné

RecherchernNumeroDonnésPointDeVente(String s)(PointDeVente)-Operateur;

// retourne l indice du numéro du point de vente:

getMemeNuméroEmplacement(String num)(int );

//-----methodes de remplissage:

//--------------------REMPLISSAGE CLIENT:

RemplissageClients()(void)-Operateur;

//-------------------REMPLISSAGE POINT DE VENTE:

RemplissagePointDeVente() ( void)-Operateur;

* Point de vente :

//Attribut

String nomAgence;

Adresse adresse;

Type typ;

String numeroTelephone

//--------------------------------------------------------------------PointDeVente()-PointDeVente;//constructeur

PointDeVente(String nomAgence, Adresse adresse, Type typ, String numeroTelephone) –Operateur; /\*si le numéro entrée est invalide une exception ssera lance\*/

//----getter et setter

//----method saisit():

Saisir()(Void)-PiontDeVente;

toString()(String)-PiontDeVente;

* Appel/SMS :
* Abonnement :

Abonnement (abstract classe) :

protected Date dateDebAbon;

protected int durée;//minute

// (solde):à consommé(cumulé pour abnnement libre / decrementer pour les autre)

protected double soldeMontant;

Abonnement()-Abonnement;//constructeur sans //parameter

Abonnement(Date dateDebAbon, int durée, double soldeMontant)-Abonnement;/\*constructeur avec paramètre : envoie deux exceptions en cas d’erreur throws DateException(date invalide), NegativeException (durée ou montant négative)\*/

abstract void Saisir();-Abonnement;

toString()(String)-Abonnement;//envoie la date debut d’abonnement

Afficher()(void)-Abonnement;//affiche la description de l’objet via sa refférence

abstract verifierAtteinteDateFinAbon()(Date);//envoie la date limite de l’abonnement

abstract verifierAtteinteDateFBloqué();//envoie la date limite de blocage de l’abonnement

//-getter et setter

//\*\*\*\*\*\*\*Méthodess pour Ajouter appel et calcul facture\*\*\*\*\*\*\*\*\*\*

isBloqué(Date d)(boolean)-Abonnement ;/\*on comparant la date donnée supposant la date actel avec la date limite de blocage si elle est atteinte ou la dépasse alors on renvoie vrai sinon faaux\*/ abstract AjouterAppel(Client client, String s)(void) -Abonnement; abstract AjouterAppel(Client client, String s, Date D,Heure h, int durée )(Appel) throws SoldeException, DateException-Abonnement ; abstract AfficherChoixAbonnement()(int);/\*affiche un menu pour choisir les differents propositions que donne l’opérateur à son client\*/

Abstract Facture(Client c):void-Abonnement; ,//afficher la facture (description)

Abonnement Libre :

//attribut ; private :

final double montantFixe=500;

final int TVA= 19;

int duréeFixe;//on en a besoin pour le renouvlement

private int duréePourBloqué;/\*cette durée est définit dans le constructeur comme étant : duréePourBloqué=durée+durée;\*/

//constructeur sans paramètre :

AbonLibre () –AbonLibre ;

//constructeur avec paramètres :

AbonLibre (Date dateDebAbon, int durée, double soldeMontant ) throws DateException, NegativeException-AbonLibre ;//ici soldeMontant est=à 0

Saisir()(void)-AbonLibre ;

Afficher()(void)-AbonLibre ;/\*appelé super.Afficher() puis donné la description de l’objet AbonLivre via sa refférence \*/

toString ()(String ) ;//envoie :  « Type Abonnement , Date Fin Abonnement , solde cumulée »

verifierSoldeRestant()(Boolean) –AbonLibre//on n’en aura pas besoin ; verifier que les dates

verifierAtteinteDateFinAbon()(Date )-AbonLibre;/\*renvoie la date après duréefixe de la (Date Debut Abonnement;\*/

Facture(Client c):void-AbonLibre;/\* affiche la date de blocage, description du client, u type d’abonnement, des listes d’appels/sms effectué reçu, et le solde à payee\*/

//ajouter appel

AjouterAppel(Client client, String s, Date D, Heure h, int durée) : Appel-AbonLibre;/\* //on envoie les trois sms entrant puis on va voir si on bloque ou non ; si la date limitée est atteinte le client ainsi ne peut que recevoir= arrive à la date de blocage le client est bloqué (ne peut ni recevoir nii emmetre des appel ) on peux le debloqué en payant la facture ; réinitialisé l’abonnement de ce client\*/

//'on decide de bloqué ou de payer; réajouter un nouvel

Abonnement Forfaitaire :

**//atributs**

**private** **int** duréeFixe;

**private** **int** duréeFixeBloqué;

**private** Double SoldeFixe;//on en aura besoin au cas de renouvellement

//constructeur sans paramètre :

AbonForfaitaire () –AbonForfaitaire ;

//constructeur avec paramètres :

AbonForfaitaire (Date dateDebAbon, int durée, double soldeMontant ) throws DateException, NegativeException-AbonLibre ;//ici soldeMontant est à > 0

Saisir() :void-AbonForfaitaire ;//on utilise ici un menu à choix (2500da, 1000da….)

Afficher() :void-AbonForfaitaire ;//

toString () : String-AbonForfaitaire ;/ \*« /envoie : -Type Abonnement ,date fin du délais, solde restant »\*/

verifierSoldeRestant()(Boolean) – AbonForfaitaire /\*on en aura besoin ; verifier que le solde est >=0\*/

verifierAtteinteDateFinAbon()(Date )- AbonForfaitaire;/\*renvoie la date après duréefixe de la (Date Debut Abonnement;\*/

Facture(Client c):void- AbonForfaitaire;/\* affiche la date de blocage, description du client, u type d’abonnement, des listes d’appels/sms effectué reçu, et le solde à payee\*/

//ajouter appel

AjouterAppel(Client client, String s, Date D, Heure h, int durée) : Appel- AbonForfaitaire /\*on envoie les trois sms entrant puis on va voir si on bloque ou non ; si la date limitée est atteinte le client ainsi ne peut que recevoir= arrive à la date de blocage le client est bloqué (ne peut ni recevoir nii emmetre des appel ) on peux le debloqué en payant la facture ; réinitialisé l’abonnement de ce client=si le solde est inssuffisant on lance une exception solde inssuffisant et on rend null\*/

Abonnement Prépayer :

Classe CarteRecharge/\*on a définit cette classe dans la classe Abonnement Prépeyer\*/

//Attribut :

Int Durée ; //mois

//-----setter et getter

//----constructeur sans paramètre

CarteRecharge() ;

CarteRecharge(**int** durée) ;

/\*Le nméro de série s’incrémente automatiquement à la fin de sa durée elle ne peut plus etre utilisé\*/

Saisir() :void/\* donnez la durée ed validitée+ le numéro de série qui doit etre valide (de longueur 14)\*/

* Autres classes supposé nécéssaire :

Date/Heure/Mois/AdresseMail : On a utilisé les classes du devoir numéro 1

Adresse : On a utilisé 3 attribut ; code rue, nomrue et wilaya ;

Classes énumération :

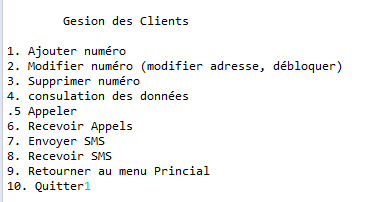
* Site – Status - Type – Wilaya – mois

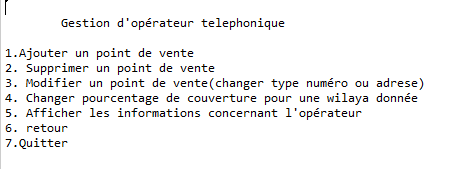
Clases Exceptions (qui hérite de Exception) :

DateException, TelephoneException, HeureException, ClientException//pour le blocage, SoldeException//pour fin de paiement ou solde == 0, SoldeInssufisantException//pour un appel qui ne s’effectue pas correctement => fin du solde

# Présntation de l’application

## programme principale





## capture d ecran

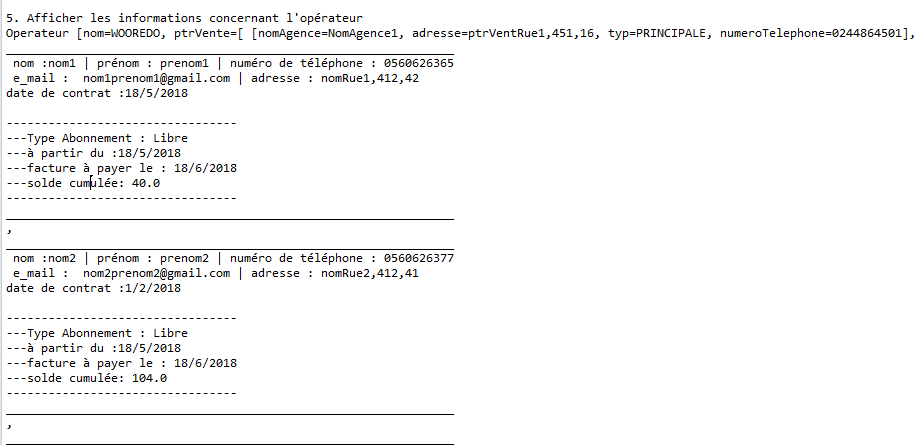


Figure - Dans la gestion de l'opérateur

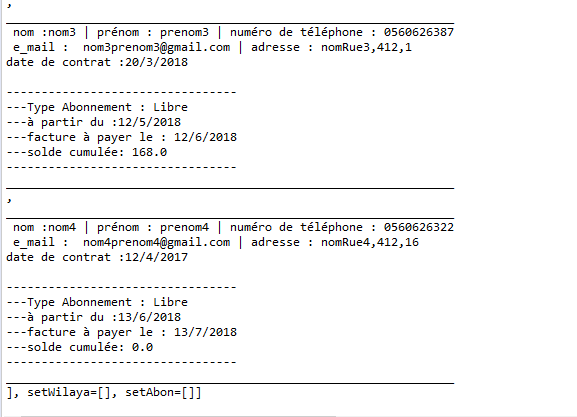
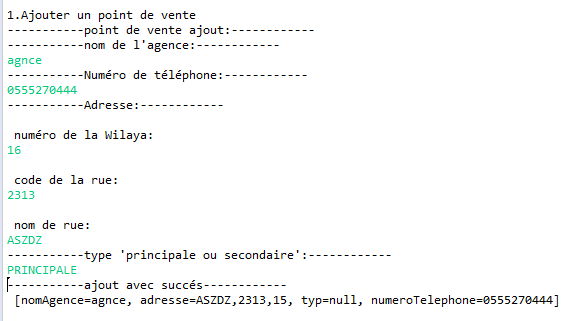


Figure 4 : suite - dans la gestion de l'opérateur- cas 1 ajouter point de vente

Figure : suite - dans la gestion de l'opérateur

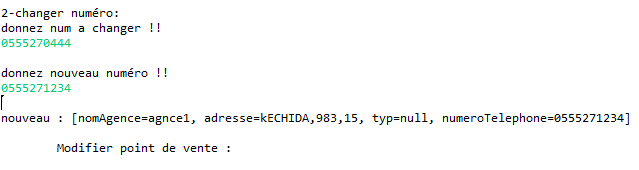
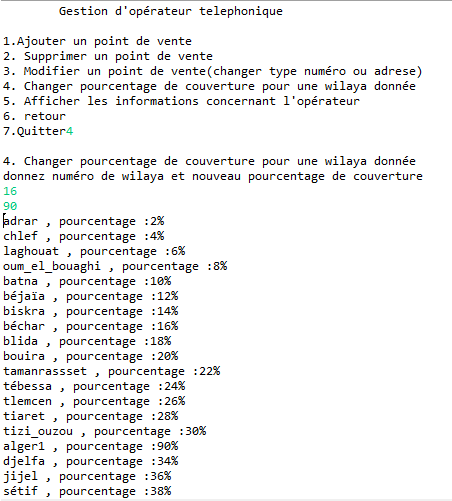


Figure 5: suite - dans la gestion de l'opérateur – dans cas 3 : Modifier un point de vente(changer type numéro ou adrese)

Figure 6: suite - dans la gestion de l'opérateur – dans cas 4 : chager le pourcentage de couverture d’une wilaya donnée

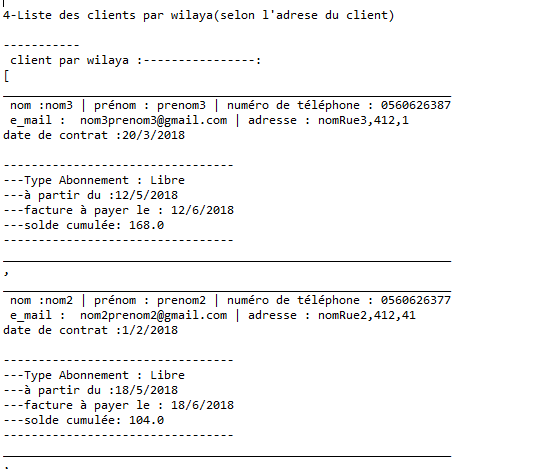
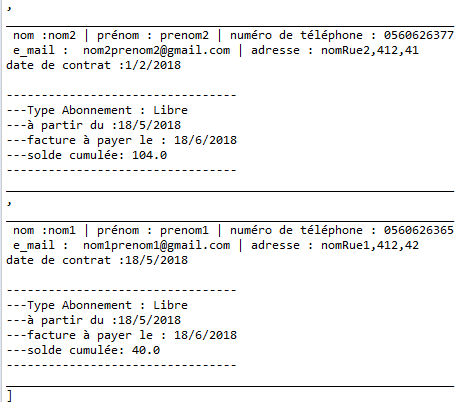


Figure 7 suite- dans la gestion des client – dans cas 4 : consultation des données-clients par wilaya

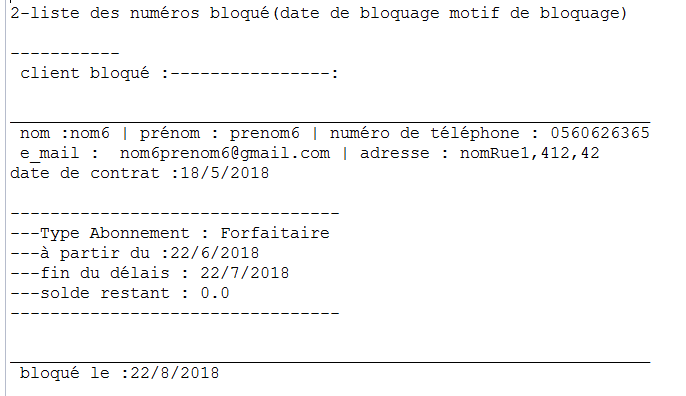


Figure 8: suite - dans la gestion des client – dans CAS 4: consultation des données- -clients bloqué

# Conclusion

On a utilisé la généricité sur les objets ceci a servi à gérer les collections. Ce concept est très utile pour développer des objets pouvant travailler avec plusieurs types de données On a utilisé la généricité sur les objets ceci a servi à gérer des collections. Ce concept est très utile pour développer des objets pouvant travailler avec plusieurs types de données ; pour les tris des collections par type d’abonnement ou wilaya définition des autres attributs liste des appels, sms, client, point de vente…).

Voir aussi la réutilisabilité du polymorphisme sans aucun risque grâce à la notion d’héritage.