De la conception UML vers son implémentation JAVA

Sommaire

Rappel de cours 1
Modèle Entity Control Boundary (ECB) 1
Singleton 1
Fabrique 1
Démarche de modélisation 2
Correspondance UML-JAVA 2
Cas étudiés 4
La vérification d'identification 4
L'enregistrement des coordonnées bancaires 4
L'identification 5
La création de profil 6
L'ajout d'aliment dans la carte du restaurant 7
La commande du client 8

Rappel de cours

Modèle Entity Control Boundary (ECB)

Notre représentation comporte :

- les « *dialogues* » ou « boundary » qui représentent les moyens d'interaction avec le système ; attention ils ne doivent comporter AUCUN objet métier,
- les « *contrôles* » ou « control » qui permettent de découpler la vue du métier.
- les « *entités* » ou « entity » qui sont les objets métiers manipulés contenant les méthodes permettant de gérer leur état.

Singleton

Le Singleton répond à deux exigences :

- garantir qu'une unique instance d'une classe donnée sera créée,
- offrir un point d'accès universel à cette instance.

Code squelette :

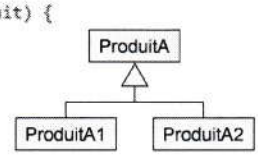
```
public class Singleton
{
    /** Constructeur privé */
    private Singleton()
    {}

    /** Holder */
    private static class SingletonHolder
    {
        /** Instance unique non préinitialisée */
        private final static Singleton instance = new Singleton();
    }

    /** Point d'accès pour l'instance unique du singleton */
    public static Singleton getInstance()
    {
        return SingletonHolder.instance;
    }
}
```

Fabrique

La fabrique permet de créer un objet dont le type dépend du contexte. L'objet retourné par la fabrique :

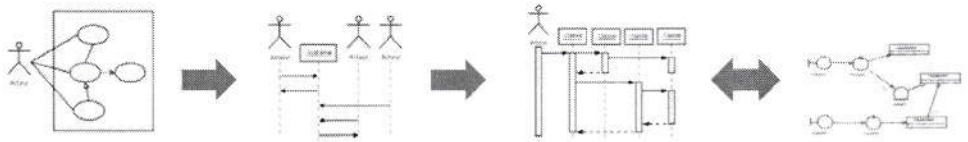
- fait partie d'un ensemble de sous-classes.
- est donc toujours du type de la classe mère. Grâce au polymorphisme les traitements exécutés sont ceux de l'instance créée.

```
public class Factory {
    public static final int TYPE_PRODUITA1 = 1;
    public static final int TYPE_PRODUITA2 = 2;

    public ProduitA getProduitA(int typeProduit) {
        ProduitA produitA = null;
        switch (typeProduit) {
        case TYPE_PRODUITA1:
            produitA = new ProduitA1();
            break;
        case TYPE_PRODUITA2:
            produitA = new ProduitA2();
            break;
        default:
            throw new IllegalArgumentException("Type de produit inconnu");
        }

        return produitA;
    }
}
```



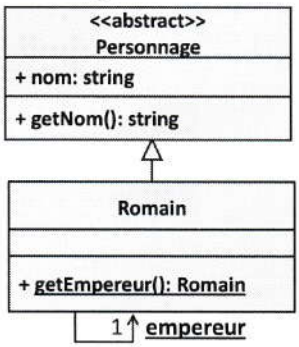
: Exemple de fabrique

Démarche de modélisation



Particularités des diagrammes :
- 1 control par cas
- 1 boundary par cas et par acteur

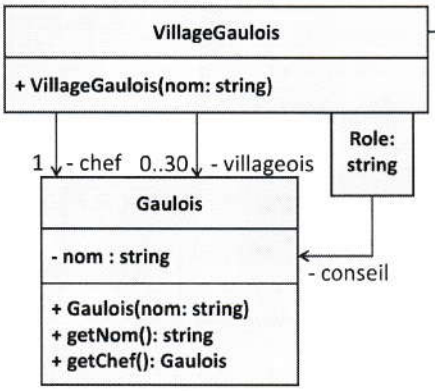
Correspondance UML-JAVA




```
public class Romain extends Personnage {
  private static Romain empereur = new Romain("César");

  public Romain(String nom){
    super(nom);
  }

  public static Romain getEmpereur() {
    return empereur;
  }
}
```

```
public abstract class Personnage {
protected String nom;

  public Personnage(String nom) {
    this.nom = nom;
  }

  public String getNom() {
    return nom;
  }
}
```

System.out.println(Romain.getEmpereur()); ➡ César



UML → Traduction JAVA
+ → public
# → protected
- → private
~ → nomAttribut → static

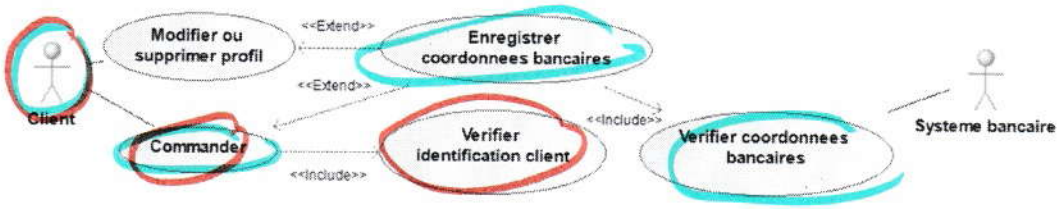
```
public class VillageGaulois extends Village {
  private Gaulois chef;
  private Gaulois[] villageois= new Gaulois[30];
  private List<Combat> combats = new ArrayList<>();
  private Map<String, Gaulois> conseil= new HashMap<>();
...
}
```

```
public interface GestionTrophee {
  void ajouterTrophee(Gaulois proprietaire,
                      Equipement trophee,
                      GradeRomain grade);
  String tousLesTrophees();
  String lesTrophees(Gaulois proprietaire);
}
```

OU


```
public class GoudurixGestion implements GestionTrophee {
  private Gaulois proprietaires[] = new Gaulois[30];
  private Equipement trophees[] = new Equipement[30];
  private GradeRomain[] grades = new GradeRomain[30];
  private int nombreDeTrophee = 0;

  public String tousLesTrophees() {
    String tousLesTrophees = "Tous les trophées du musée sont :\n";
    for (int i = 0; i < nombreDeTrophee; i++)
      tousLesTrophees += "- " + trophees[i] + "\n";
    return tousLesTrophees;
  }
  ...
```

Cas inclus / étendu



Cas « vérifier identification client » ou « vérifier coordonnées bancaires ».
Dans ces deux cas il n'y a pas d'interaction avec un acteur « humain ».

La méthode *verifierIdentificationClient*, point d'entrée du cas « vérifier identification Client », va regarder si un attribut correspondant à un booléen permettant de savoir si l'utilisateur est connecté ou non dans l'entité client est bien à **true**.
Mais il n'y aura pas d'interaction avec l'acteur HUMAIN client.

La méthode *verifierCoordonneesBancaires*, point d'entrée du cas « vérifier coordonnées bancaires », va correspondre avec un acteur NON HUMAIN pour savoir si les informations concernant une carte bancaire sont valides. Cet acteur non humain est un logiciel extérieur au logiciel en cours de développement chez burgerResto correspondant à un logiciel traitant les informations bancaires entre des applications et les banques.

Dans ces deux cas les contrôleurs des cas principaux vont échanger des messages avec les cas inclus ou étendus :

- le contrôleur du cas «commander » va appeler la méthode *verifierIdentificationClient* du contrôleur du cas « identification Client »,
- le contrôleur du cas «enregistrer coordonnées bancaires » va appeler la méthode *verifierCoordonneesBancaires* du contrôleur du cas « vérifier coordonnées bancaires ».

Cas « enregistrer coordonnées bancaires ».
Lorsqu'un profil est créé il n'a pas de carte bancaire qui lui est associé, seulement un nom, un prénom, un login et un mot de passe.

Si on regarde le diagramme de cas, le client peut enregistrer ses coordonnées bancaires :

- lors du cas « Modifier ou supprimer profil »,
- lors du cas « commander ».

Mais le client ne peut pas le faire indépendamment d'un de ces deux cas.

Par contre, contrairement au cas « vérifier identification », le cas « enregistrer coordonnées bancaires » a besoin de l'acteur « client » afin qu'il entre le numéro et la date d'expiration de la carte bancaire.
Un boundary ne pouvant communiquer qu'avec un contrôleur ou un autre boundary, c'est le boundary du cas principal (« commander » ou « modifier ou supprimer profil ») qui appellera le boundary du cas étendu (ou dans un autre exemple inclus) « enregistrer coordonnees bancaires ».

Les erreurs à ne pas commettre
Conceptuellement IL NE FAUT PAS :

- Faire des interactions avec un acteur humain depuis un contrôleur,
- Mettre un boundary comme attribut d'un contrôleur.

A retenir

Un cas inclus ou étendu doit échanger avec un acteur humain => le boundary du cas principal appelle la méthode d'entrée du boundary correspondant au cas inclus ou étendu.

Un cas inclus ou étendu ne doit pas échanger avec un acteur humain => le contrôleur du cas principal appellera la méthode d'entrée du contrôleur correspondant au cas inclus ou étendu.

En conséquence :
Les constructeurs des boundary prennent en paramètres d'entrée :

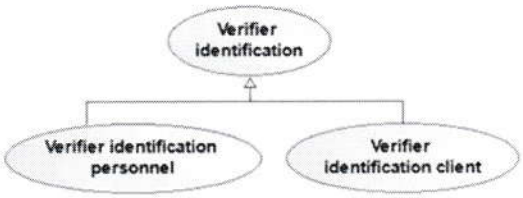
- **Le contrôleur du cas,**
- **Les boundary des cas inclus et/ou étendu qui ont une interaction avec l'acteur.**

Les contrôleurs ne possèdent pas de constructeur sauf si le cas principal contient des cas inclus et/ou étendu qui n'ont pas d'interaction avec l'acteur, le contrôleur du cas principal prend en paramètres d'entrée les contrôleurs des cas inclus et/ou étendu.

Cas étudiés

La vérification d'identification

Diagramme de cas « Vérifier Identification »



Travail à effectuer

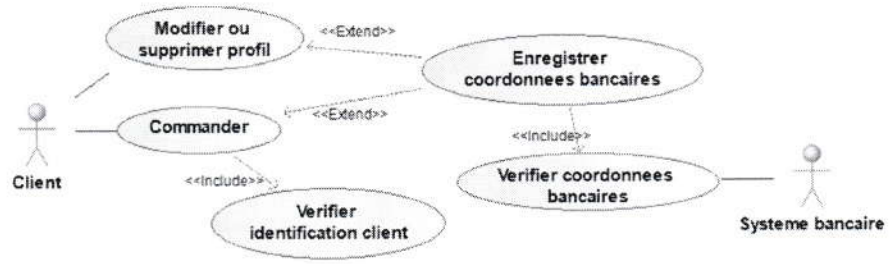
Donner les diagrammes de séquence détaillés des cas correspondant à la vérification d'identification.
En parallèle construire le diagramme de classes participantes à ces cas.
Implémenter votre modélisation.

Attention : votre modélisation devra obligatoirement :

- respecter le framework ECB,
- comporter :
  - la classe BDClient contenant une Map associant le numéro du client à l'objet du type Client,
  - la classe BDPersonnel contenant une Map associant le numéro de l'employé à l'objet du type Personnel,
  - un seul control pour les deux cas de création de vérification d'identification.
- Implémenter les classes BDClient et BDPersonnel comme des singletons

L'enregistrement des coordonnées bancaires

Diagramme de cas « Enregistrer coordonnées bancaires »



Travail à effectuer

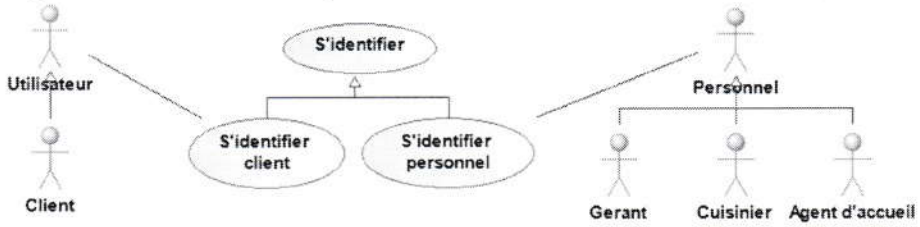
Donner le diagramme de séquence détaillé du cas correspondant à la l'enregistrement des coordonnées bancaires.
En parallèle construire le diagramme de classes participantes à ce cas.
Implémenter votre modélisation.

Attention : votre modélisation devra obligatoirement :

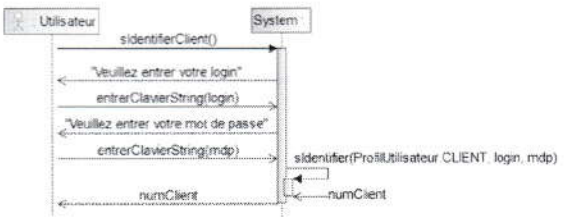
- respecter le framework ECB,
- la classe BDClient contenant une Map associant le numéro du client à l'objet du type Client,
- Implémenter la classe BDClient comme singleton.

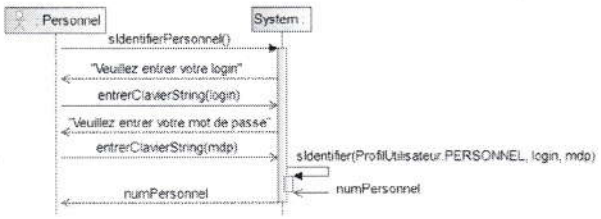
L'identification

Diagramme de cas « S'identifier »



Diagrammes de séquence système « S'identifier »





Travail à effectuer

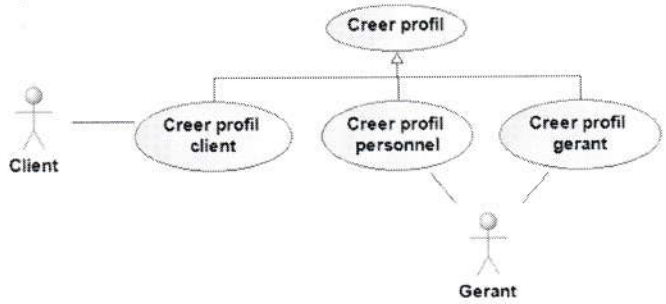
Donner les diagrammes de séquence détaillés des cas correspondant à l'identification.
En parallèle construire le diagramme de classes participantes à ces cas.
Implémenter votre modélisation.

Attention : votre modélisation devra obligatoirement :

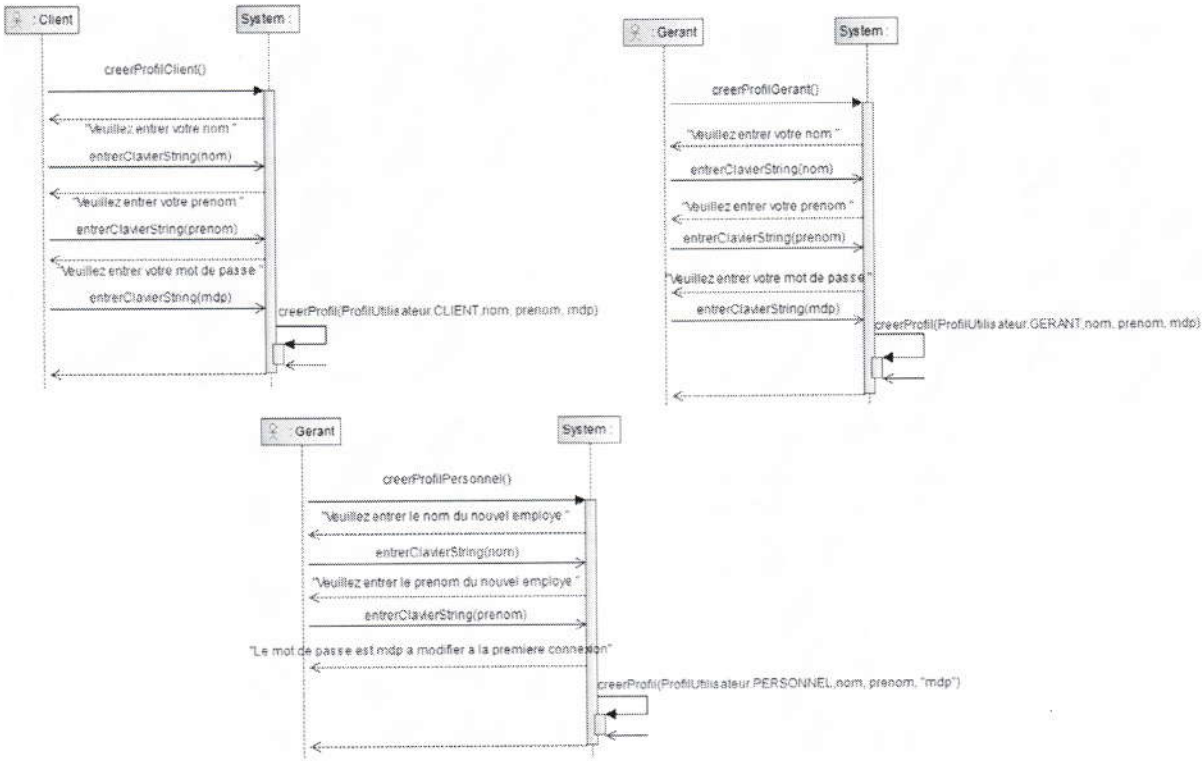
- respecter le framework ECB,
- comporter :
  - la classe BDClient contenant une Map associant le numéro du client à l'objet du type Client,
  - la classe BDPersonnel contenant une Map associant le numéro de l'employé à l'objet du type Personnel,
  - un seul control pour les deux cas d'identification.
- Implémenter les classes BDClient et BDPersonnel comme des singletons

La création de profil

Diagramme de cas « Créer Profil »



Diagrammes de séquence système « Créer Profil »



Travail à effectuer

Donner les diagrammes de séquence détaillés des cas correspondant à la création de profil.
En parallèle construire le diagramme de classes participantes à ces cas.
Implémenter votre modélisation.

Attention : votre modélisation devra obligatoirement :

- respecter le framework ECB,
- comporter :
  - la classe BDClient contenant une Map associant le numéro du client à l'objet du type Client,
  - la classe BDPersonnel contenant une Map associant le numéro de l'employé à l'objet du type Personnel,
  - un seul control pour les deux cas de création de profil,
  - la classe FabriqueProfil comme fabrique des différents comptes (client, personnel, gérant).
- Implémenter les classes BDClient et BDPersonnel comme des singletons.

L'ajout d'aliment dans le menu du restaurant

Diagramme de cas « Ajouter aliment menu »

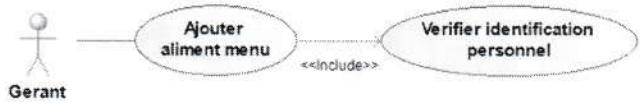
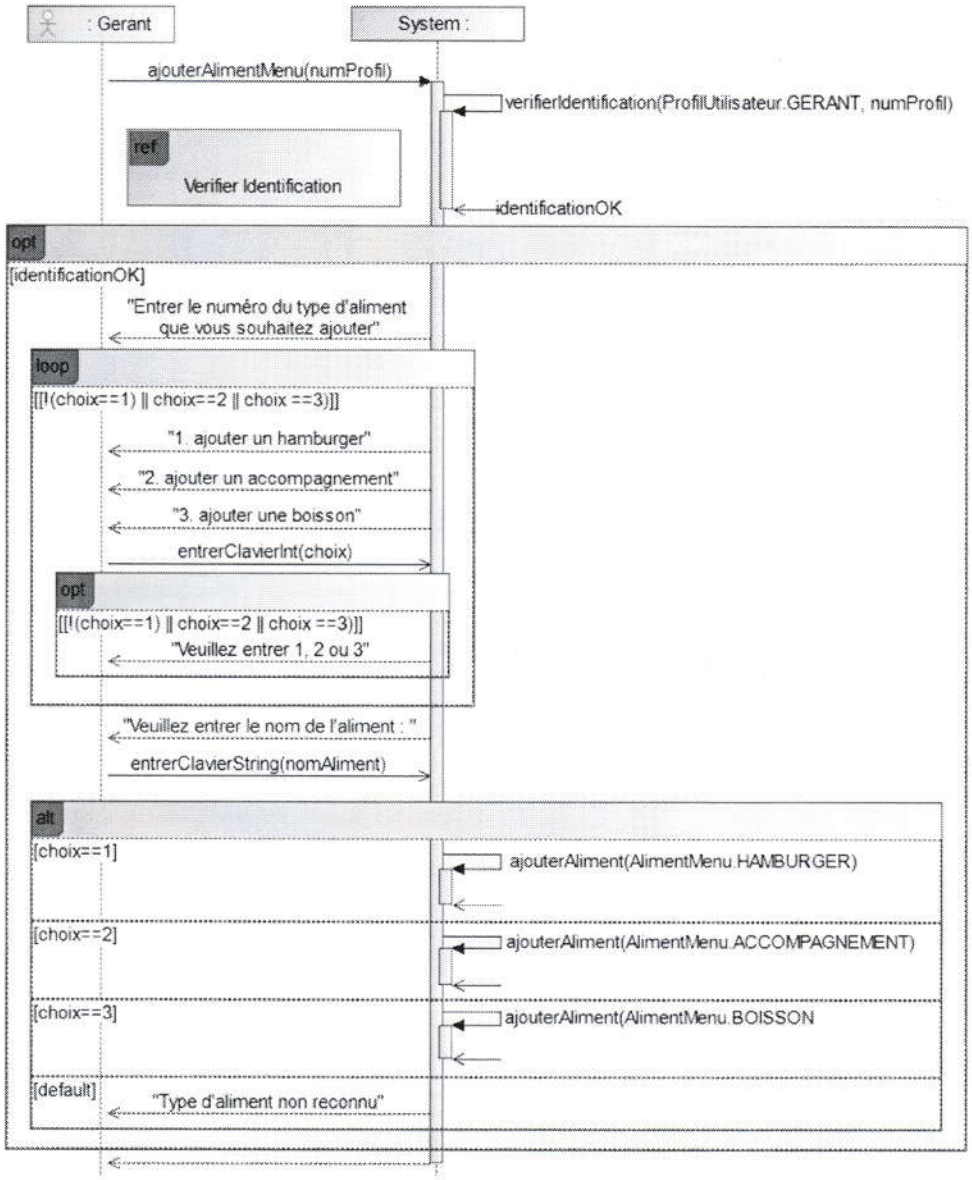


Diagramme de séquence système « Ajouter aliment menu »



Travail à effectuer

Donner les diagrammes de séquence détaillés des cas correspondant à la mise à jour de la carte.
En parallèle construire le diagramme de classes participantes à ce cas.
Implémenter votre modélisation.

Attention : votre modélisation devra obligatoirement :

- respecter le framework ECB,
- comporter La classe FabriqueAliment comme fabrique des différents types d'aliments possibles (hamburger, accompagnement, boisson).
- Implémenter la classe Carte comme des singletons contenant 3 ArrayList : listeHamburger, listeAccompagnement, listeBoisson.

La commande du client

Diagramme de cas « Commander »

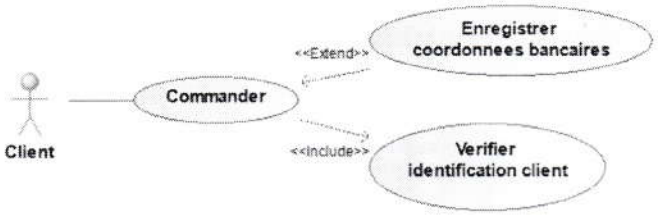
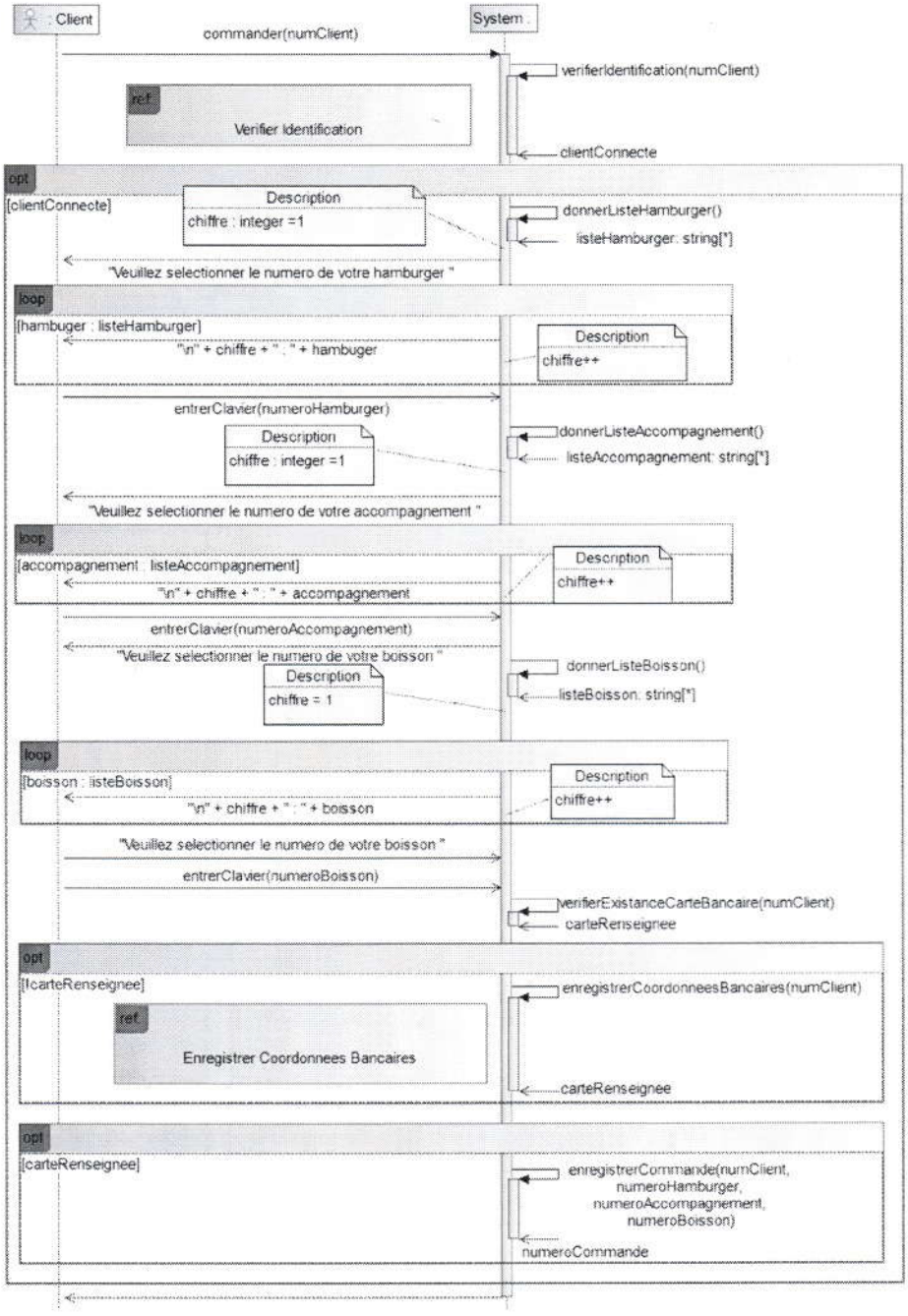


Diagramme de séquence système « Commander »



Travail à effectuer

Donner le diagramme de séquence détaillé du cas correspondant à prise d'une commande d'un client.
En parallèle construire le diagramme de classes participantes à ce cas.
Implémenter votre modélisation.

Attention : votre modélisation devra obligatoirement :

- respecter le framework ECB,
- la classe BDClient contenant une Map associant le numéro du client à l'objet du type Client,
- Implémenter les classes BDClient et BDPersonnel comme des singletons.

La commande du client

Diagramme de cas « Commander »

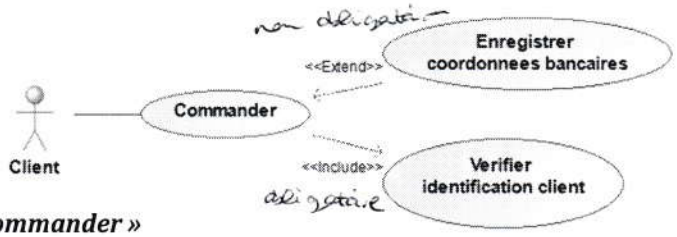
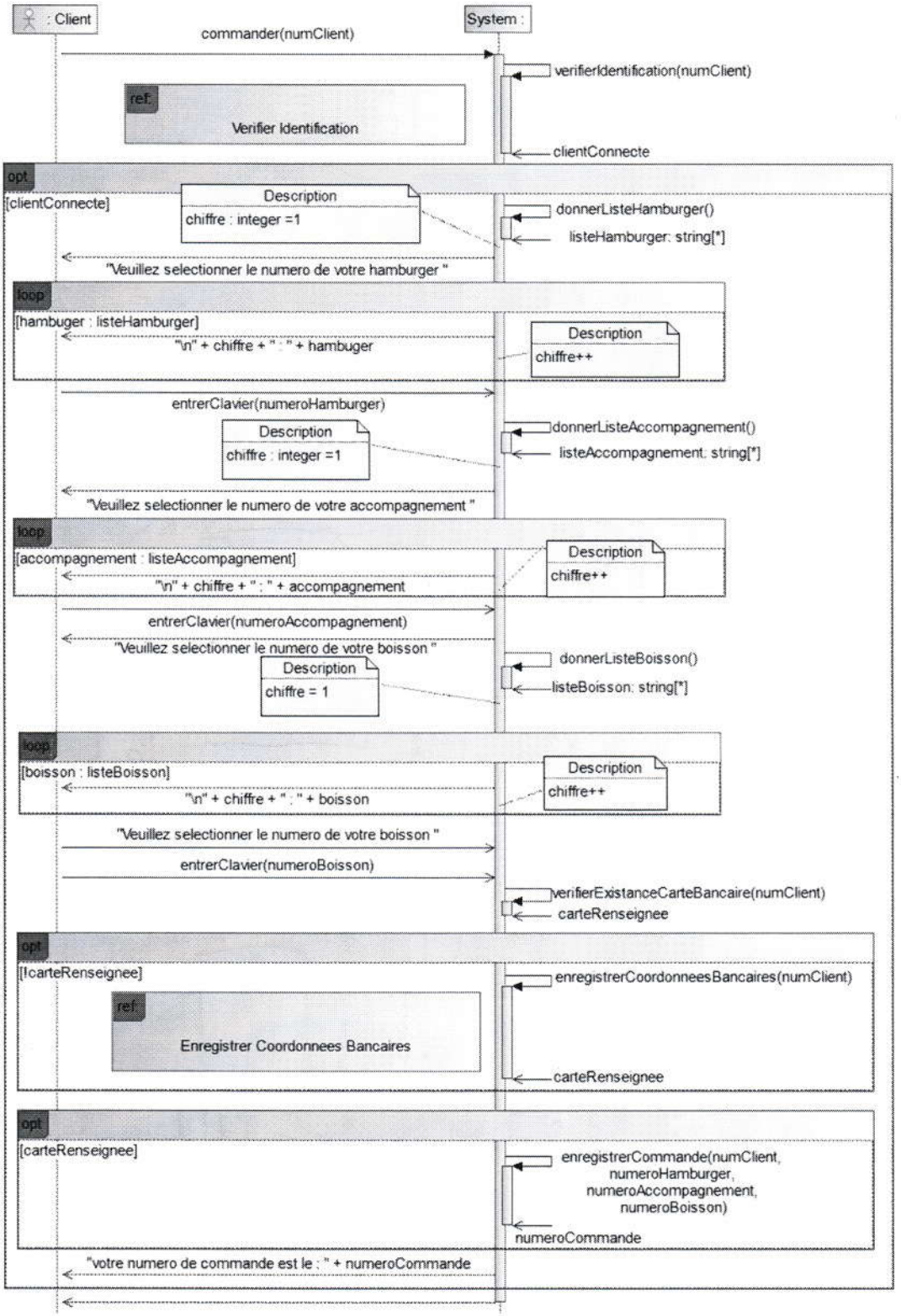
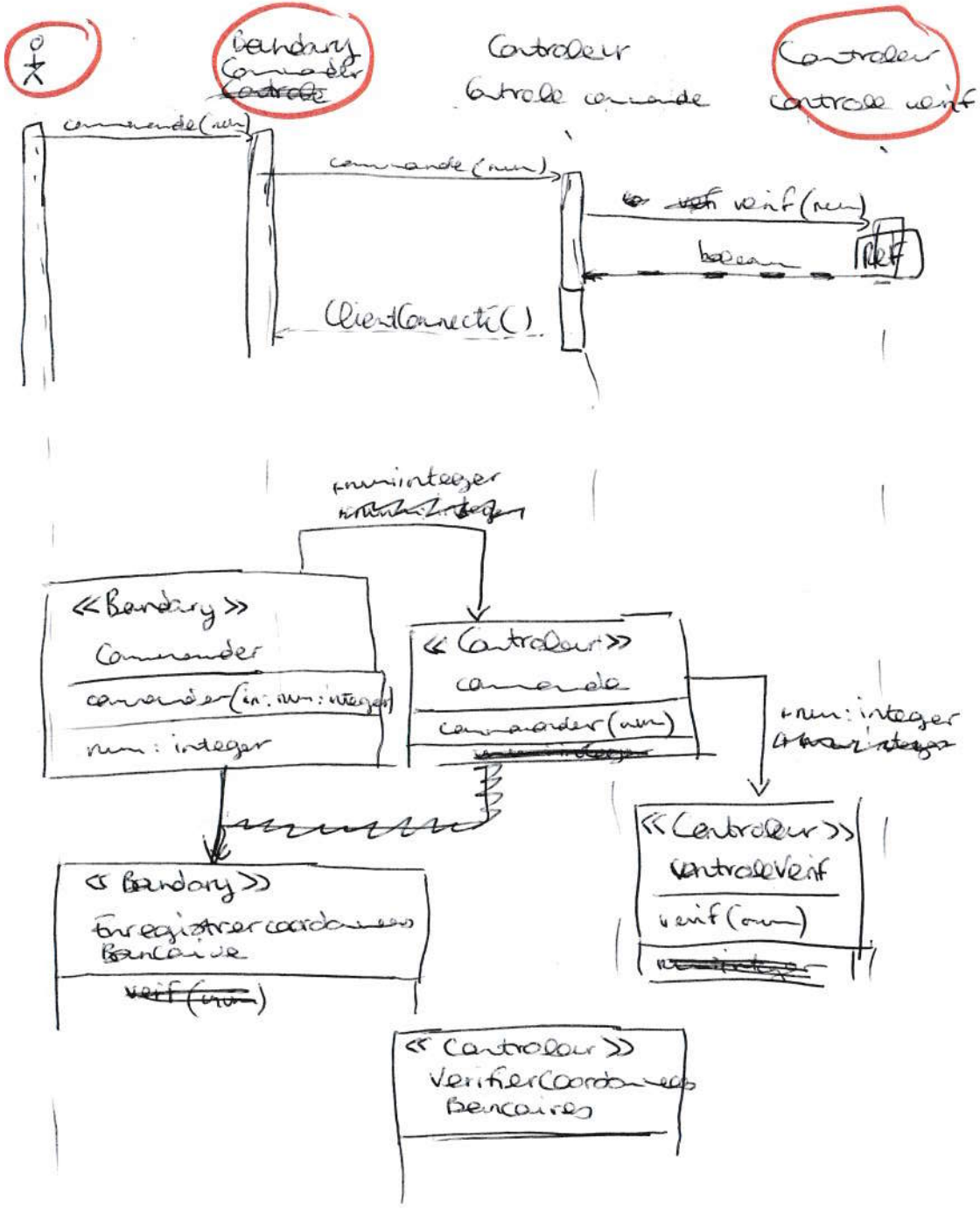
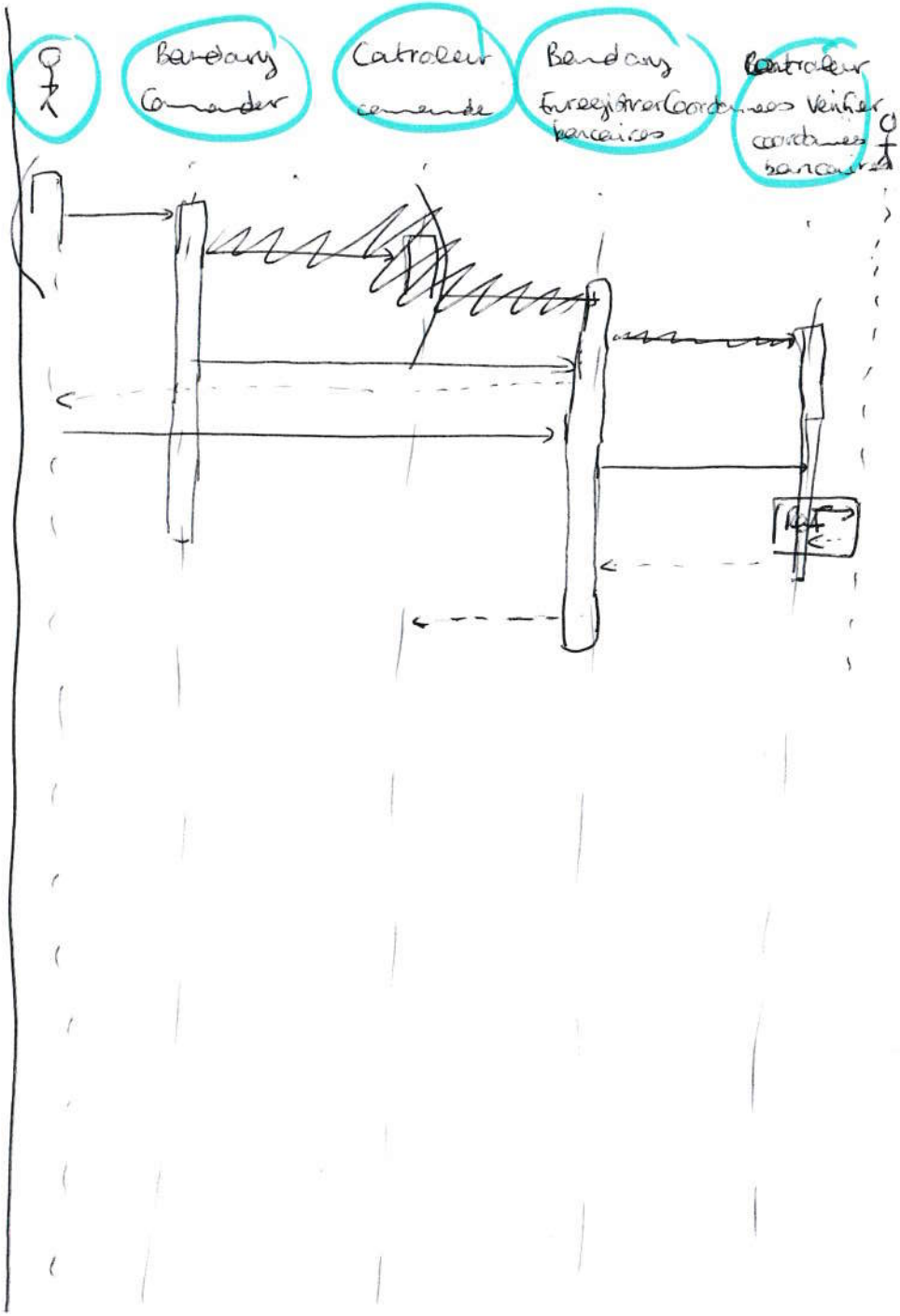


Diagramme de séquence système « Commander »




Boundary Commander
Controleur
Controleur
commande(num)
commande(num)
veif(num)
bolean
ClientConnecté()
+num:integer
«Boundary»
Commander
commander(in: num: integer)
num: integer
«Controleur»
commande
commander(num)
+num: integer
«Controleur»
controleVeif
veif(num)
«Boundary»
Enregistrer coordonnées Bancaire
«Controleur»
VerifierCoordonnées Bancaires


Boundary Commander
Controleur
commande
Boundary
EnregistrerCoordonnées bancaires
Controleur
Verifier coordonnées bancaires