# Projet CSC4102 : Gestion des clefs dans un hôtel

LE GLAUNEC Alexis &  CANTO Guillem
Année 2019–2020 — 17 janvier 2020

# Table des matières

# 1 Spécification

## 1.1 Diagrammes de cas d’utilisation

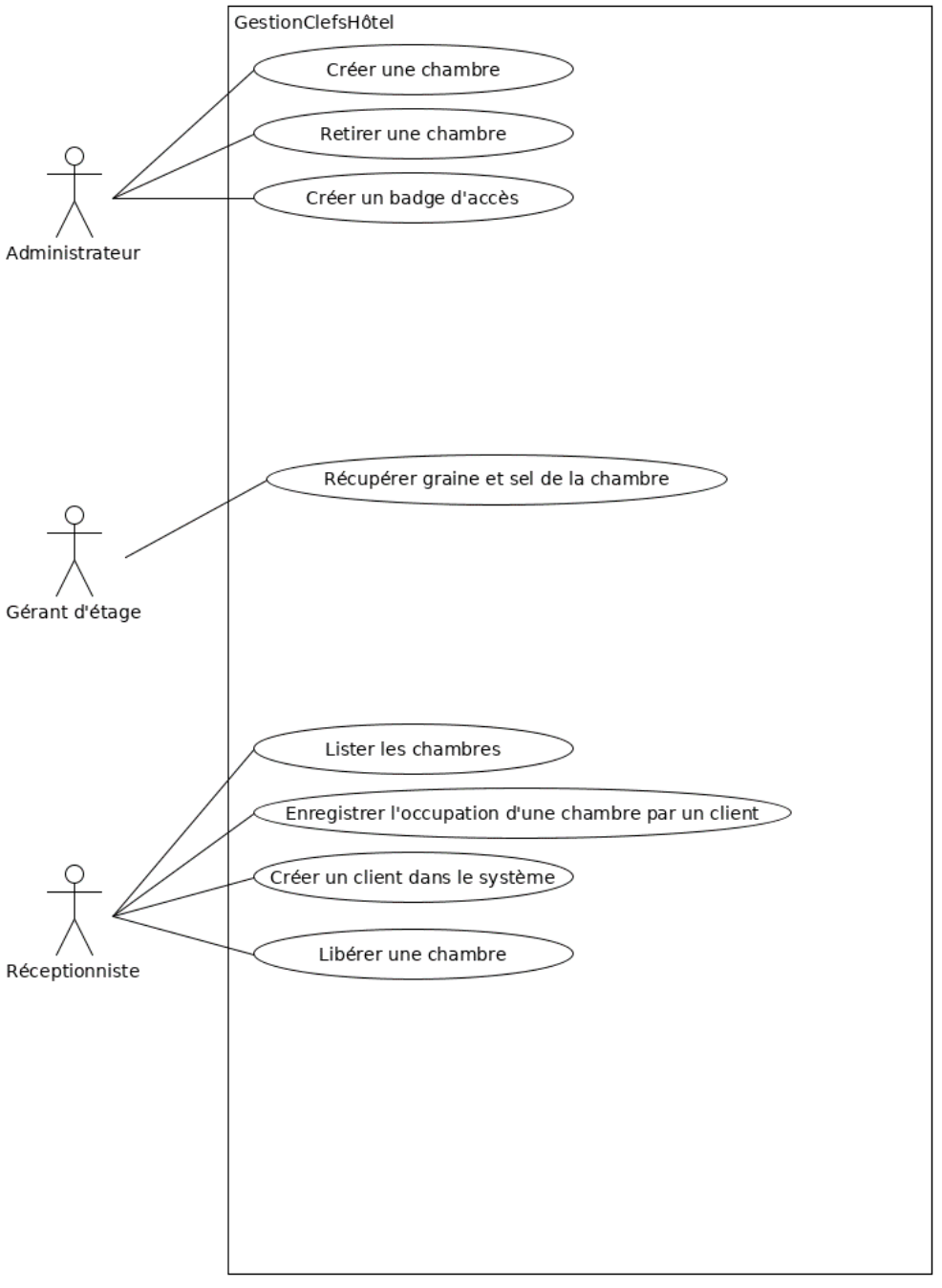


FIG. 1: Diagramme de cas d’utilisation

## 1.2 Priorités, préconditions et postconditions des cas d'utilisation

- Créer une chambre : **priorité haute**
  - précondition : identifiant/code de la chambre bien formé (non null et non vide) ∧ chambre avec ce code inexistante ∧ graine pour la génération des clefs bien formée (non null et non vide)
  - postcondition : chambre avec ce code existant
- Libérer une chambre : **priorité haute**
  - précondition : identifiant/code de la chambre bien formé (non null et non vide) ∧ chambre avec ce code existante ∧ chambre précédemment non libre
  - postcondition : clés sur le badge associé effacées ∧ badge à la réception ∧ client n'occupe plus de chambre ∧ chambre libre
- Enregistrer l'occupation d'une chambre par un client : **priorité haute**
  - précondition : identifiant/code du client bien formé (non null et non vide) ∧ identifiant/code de la chambre bien formé (non null et non vide) ∧ identifiant du client dans le système ∧ chambre avec ce code existante ∧ chambre avec ce code libre ∧ client n'occupe pas de chambre ∧ badge à la réception ∧ identifiant/code du badge bien formé (non null et non vide)
  - postcondition: chambre avec ce code occupée ∧ badge prêté au client ∧ client occupe une chambre
- Créer un badge d'accès : **priorité haute**
  - précondition : identifiant/code du badge bien formé (non null et non vide) ∧ badge avec cet identifiant inexistant
  - postcondition: badge avec cet identifiant existant
- Créer un client dans le système : **priorité haute**
- Récupération d'une graine et d'un sel : **priorité haute**
- Retirer une chambre : **priorité basse**
- Lister les chambres : **priorité moyenne**

# 2 Préparation des tests de validation

## 2.1 Tables de décision des tests de validation

| Numéro de test | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Identifiant/code de la chambre bien formé (≠ null ∧ ≠ vide) | F | T | T | T |
| Graine pour la génération des clefs bien formée (≠ null ∧ ≠ vide) | | F | T | T |
| Chambre inexistante avec ce code | | | F | T |
| Création acceptée | F | F | F | T |
| Chambre avec ce code existant | F | F | F | T |
| Nombre de jeux de test | 2 | 2 | 1 | 2 |

TAB. 1: Cas d'utilisation « créer une chambre »

| Numéro de test | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| identifiant/code du client bien formé (non null et non vide) | F | T | T | T | T | T | T | T | T |
| identifiant/code de la chambre bien formé (non null et non vide) | | F | T | T | T | T | T | T | T |
| identifiant du client dans le système | | | F | T | T | T | T | T | T |
| chambre avec ce code existante | | | | F | T | T | T | T | T |
| chambre avec ce code libre | | | | | F | T | T | T | T |
| client n'occupe pas de chambre | | | | | | F | T | T | T |
| badge à la réception | | | | | | | F | T | T |
| identifiant/code du badge bien formé (non null et non vide) | | | | | | | | F | T |
| Enregistrement réussi (occupation créée) | F | F | F | F | F | F | F | F | T |
| Chambre avec ce badge occupée (occupation de chambre non null) | | | | | | | | | T |
| Badge prêté au client (occupation de badge non null) | | | | | | | | | T |
| Client occupe une chambre (occupation de client non null) | | | | | | | | | T |
| Nombre de jeux de test | 2 | 2 | 2 | 1 | 1 | 1 | 1 | 2 | 4 |

TAB. 2: Cas d'utilisation «enregistrer l'occupation d'une chambre par un client»

| Numéro de test | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| identifiant/code du client bien formé (non null et non vide) | F | T | T | T | T |
| chambre avec ce code existante | | F | T | T | T |
| chambre précédemment non libre | | | F | T | T |
| badge rendu au réceptionniste | | | | F | T |
| Libération acceptée | F | F | F | F | T |
| clés sur le badge associé effacées | | | | | T |
| badge à la réception | | | | | T |
| client n'occupe plus de chambre | | | | | T |
| chambre libre | | | | | T |
| Nombre de jeux de test | 2 | 1 | 1 | 1 | 5 |

TAB. 3: Cas d'utilisation «libérer une chambre»

# 3 Conception

## 3.1 Liste des classes

À la suite d'un parcours des diagrammes de cas d'utilisation et d'une relecture de l'étude de cas, voici une première liste de classes avec quelques attributs :

- GestionClefsHotel (la façade),
- Chambre — id, graine, sel, Occupation
- Util (classe déjà programmée) — attribut de classe TAILLE_CLEF, et méthodes de classes genererUneNouvelleClef et clefToString.
- Badge — id, clef1 (byte[]), clef2 (byte[]), Occupation
- Occupation — id, , dateDebut, dateFin,
- Client — id, Nom, Prénom, Occupation

## 3.2 Diagramme de classes

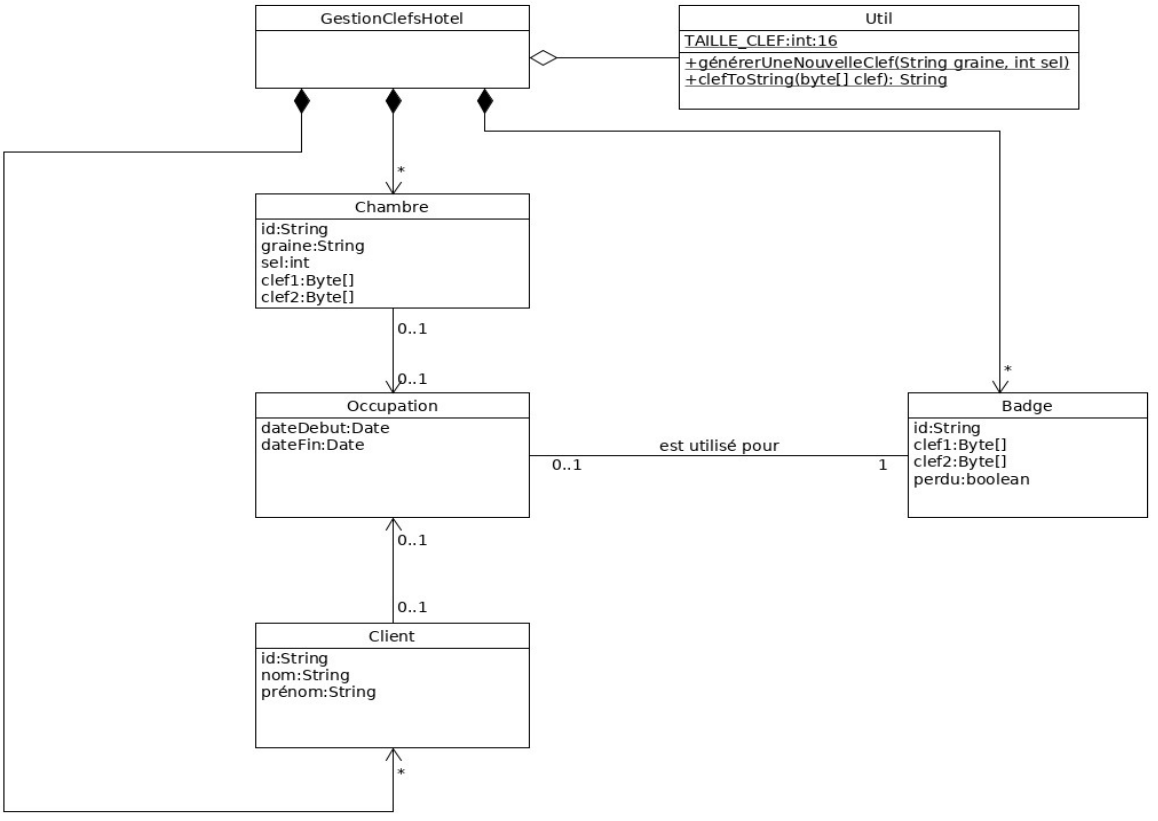


**FIG. 1: Diagramme de classes**

# 3. Diagrammes de séquence

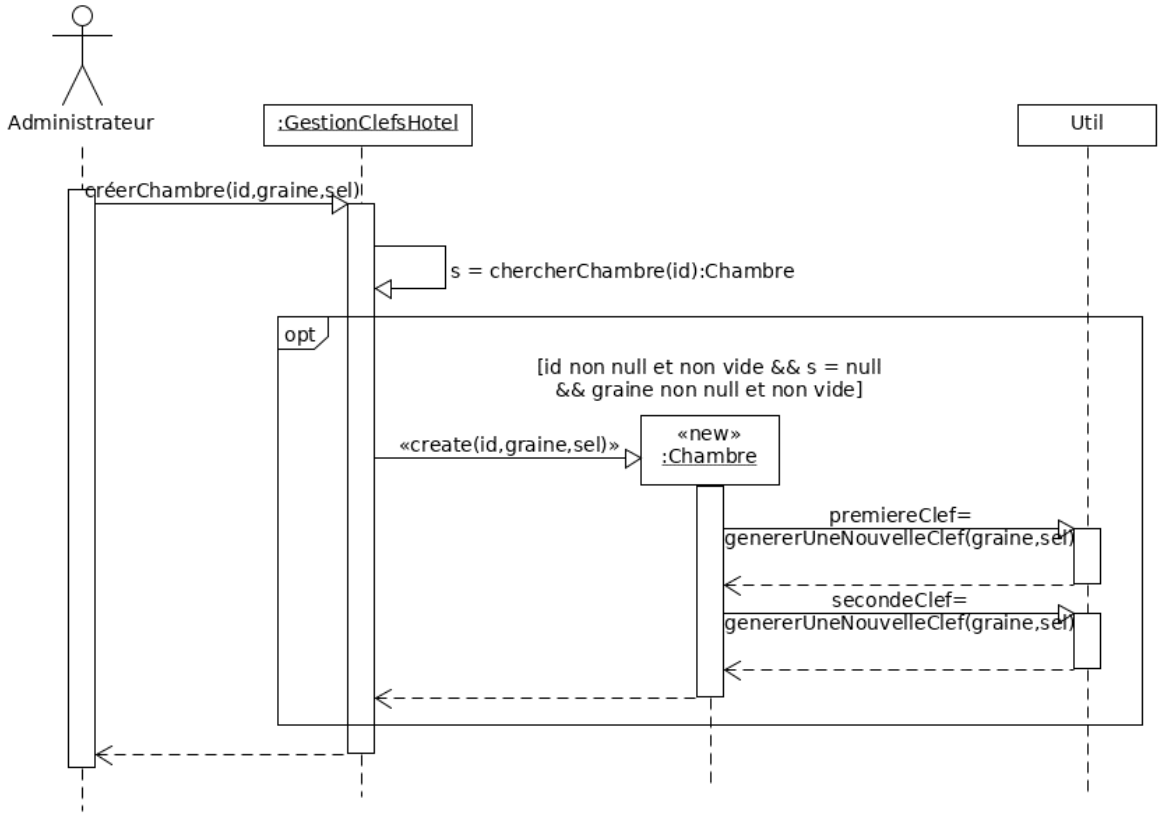


**FIG. 1: Cas d'utilisation «créer une chambre»**

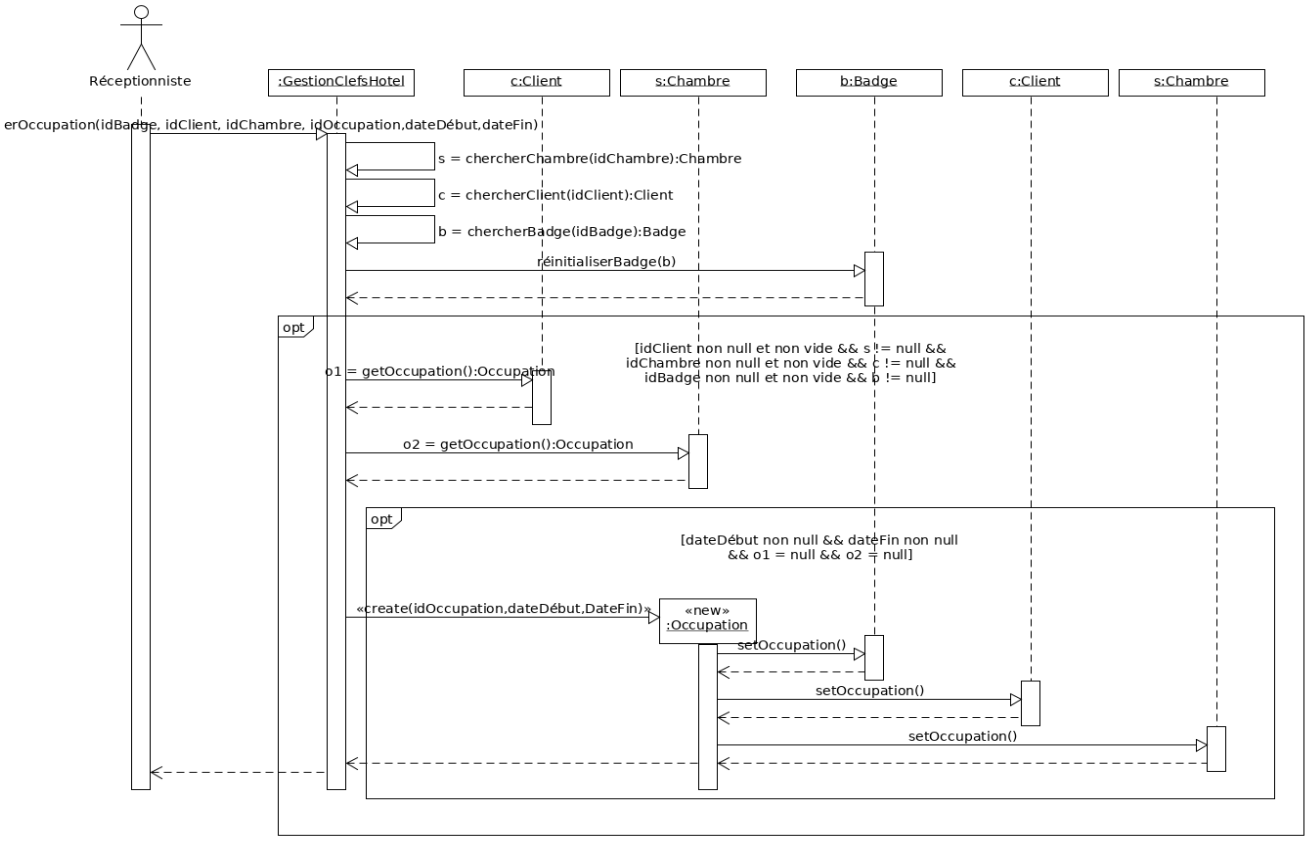


**FIG. 2: Cas d'utilisation «enregistrer l'occupation d'une chambre par un client»**

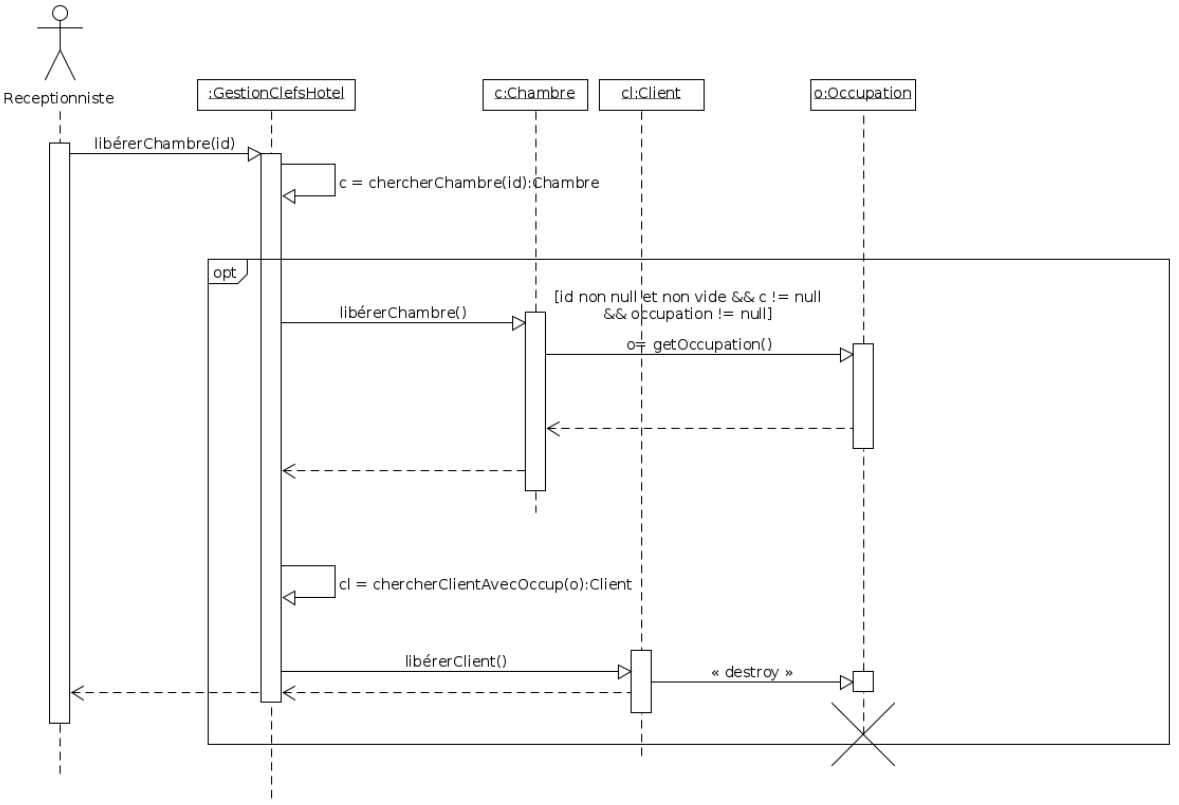


**FIG. 3: Cas d'utilisation «libérer une chambre»**

# 4 Fiche des classes

## 4.1 Classe GestionClefsHotel

| GestionClefsHotel |
|---|
| <– attributs « association » –><br>− chambres : collection de @Chambre<br>− clients : collection de @Client<br>− badge : collection de @Badge |
| <– constructeur –><br>+ GestionClefsHotel()<br><– operations « cas d'utilisation » –><br>+ créerChambre(String code, String graine, int sel)<br>+ retirerChambre(String code) |
| + créerClient(String Nom, String Prenom)<br>+ listerChambre()<br>+ créerOccupation(String idBadge, String idChambre, String idClient, DateTime dateDébut, DateTime dateFin)<br>+ récupérerGraineSelChambre(String idChambre)<br>+ créerBadge(String code)<br>+ libérerChambre(String idChambre)<br>+ getChambre(String idChambre)<br>+ getClient(String idClient)<br>+ getOccupation(String idChambre)<br>+ getOccupation(String idClient)<br>+ getBadge(String idBadge) |

## 4.2 Classe Badge

| Badge |
|---|
| <– (private) attributs « association » –><br>− id : String<br>− clef1 : tableau de Bytes<br>− clef2 : tableau de Bytes |
| − occupation : @Occupation |
| <– (public) constructeur –><br>+ Badge(String identifiant) |
| <– operations « cas d'utilisation » –> |
| + getOccupation() : Occupation |
| + estDonnéAUnClient(String idOccupation) |
| + rendreBadge(String idBadge) |
| + setClef1(String valeur) |
| + setClef2(String valeur) |
| + getClef2() |
| + invariant() : Booléen |

# 5 Diagrammes de machine à états et invariants

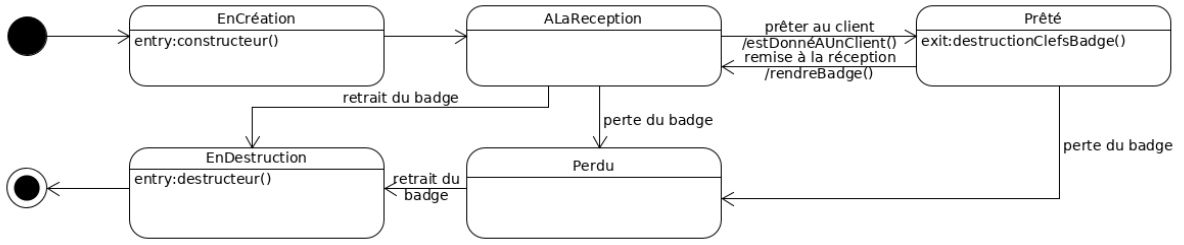


**FIG. 1: Diagramme de machine à états «Badge d'accès»**

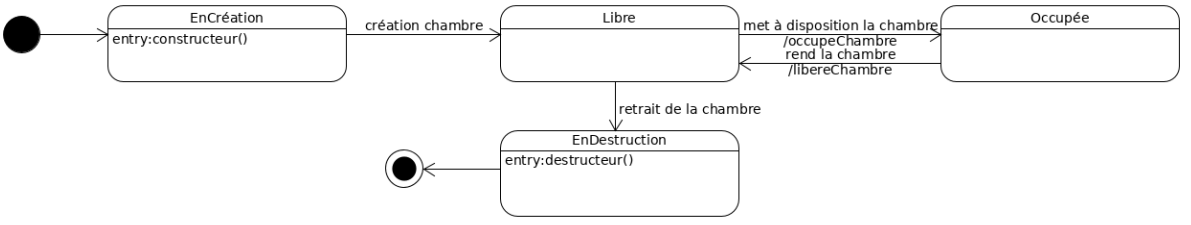


**FIG. 2: Diagramme de machine à états «Chambre»**

# 6 Préparation des tests unitaires

## 6.1 Classe Chambre

Invariant : (occupation == null ) ∨ (occupation != null)

## 6.2 Classe Badge

Invariant : ( (occupation != null) ∧¬perdu) ∨ ( (occupation == null) ∧ ¬perdu) ∨
( (occupation == null) ∧ perdu) ∧ (clef1!= null) ∧ (clef1.length == Util.TAILLE_CLEF) ∧
(clef2!= null) ∧ (clef2.length == Util.TAILLE_CLEF)

| Numéro de test | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| chambre != null | F | T | T | T | T |
| clef1' = clef2 | F | F | T | T | T |
| clef2' ≠ clef2 | F | F | F | T | T |
| ch.getSel()' = ch.getSel() + 1 | F | F | F | F | T |
| invariant | T | T | T | T | T |
| Levée exception | OUI | NON | NON | NON | NON |
| Objet crée | T | T | T | T | T |
| Nb jeux de test | 1 | 1 | 1 | 1 | 1 |

Tableau 1 : Méthode « estDonnéAUnClient » de la classe Badge

| Numéro de test | 1 | 2 | 3 |
|---|---|---|---|
| identifiant ≠ null ∧ ¬vide | F | T | T |
| occupation == null | F | F | T |
| invariant | | T | T |
| Levée exception | OUI | OUI | NON |
| Objet crée | F | F | T |
| Nb jeux de test | 2 | 1 | 1 |

Tableau 2 : Méthode « constructeur» de la classe Badge

# Annexe : Algorithmes Badge

## 1 ALGO1C1 : constructeur

badge(String identifiant)

si identifiant == null ou identifiant = "" alors lever une exception
si occupation!= null alors lever une exception
id := identifiant
perdu := false
assert invariant()

## 2 ALGO2C1 : estDonnéAUnClient

estDonnéAUnClient(Chambre ch)

si chambre == null alors lever une exception
clef1 := Clef2
clef2:= générerUneNouvelleClef(ch.graine, ch.sel + 1)
assert invariant()