Projet CSC4102 : Gestion des clefs dans un hôtel

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

Table of Contents

1 Spécification

1.1 Diagrammes de cas d’utilisation

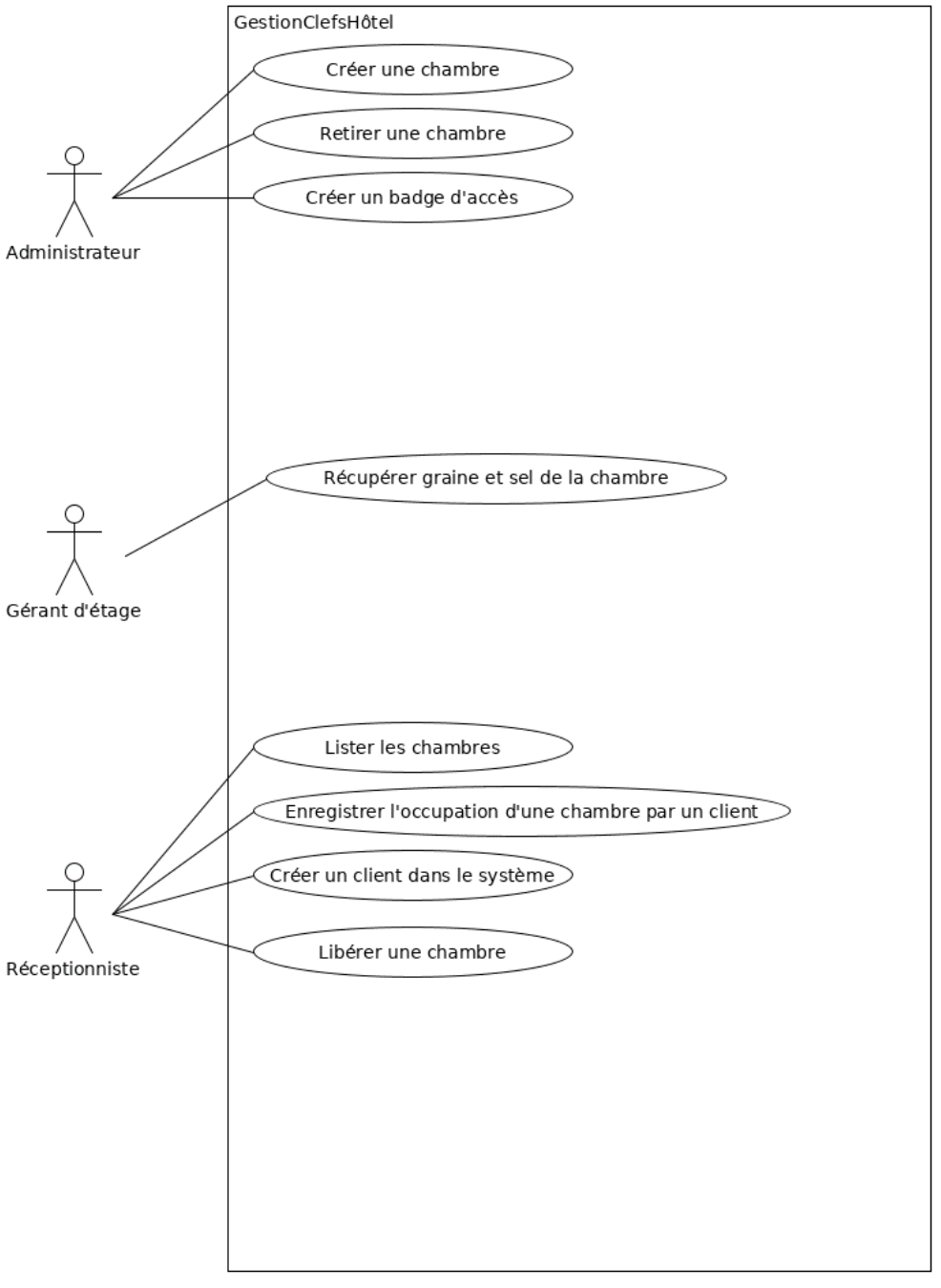

FIG. 1: Diagramme de cas d’utilisation

1.2 Priorités, préconditions et postconditions des cas d'utilisation

Les priorités des cas d'utilisation pour le sprint 1 sont choisies avec les règles de bon sens suivantes :

- pour retirer une entité du système, elle doit y être. La priorité de l'ajout est donc supérieure ou égale à la priorité du retrait ;
- pour lister les entités d'un type donné, elles doivent y être. La priorité de l'ajout est donc supérieure ou égale à la priorité du listage ;
- il est *a priori* possible, c.-à-d. sans raison contraire, de démontrer la mise en œuvre d'un sous-ensemble des fonctionnalités du système, et plus particulièrement la prise en compte des principales règles de gestion, sans les retraits ou les listages.
- la possibilité de lister aide au déverminage de l'application pendant les activités d'exécution des tests de validation.

Par conséquent, les cas d'utilisation d'ajout sont *a priori* de priorité « haute », ceux de listage de priorité « moyenne », et ceux de retrait de priorité « basse ».

Dans la suite, nous donnons les préconditions et postconditions pour les cas d'utilisation de priorité « Haute ». Pour les autres, nous indiquons uniquement leur niveau de priorité.

- 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 : clé sur le badge associé effacée ∧ 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
  - postcondition: chambre avec ce code occupée ∧ badge prêté au client ∧ client occupe une chambre ∧ identifiant/code du badge bien formé (non null et non vide)
- 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

La fiche programme du module CSC4102 ne permettant pas de développer des tests de validation couvrant l'ensemble des cas d'utilisation de l'application, les cas d'utilisation choisis sont de priorité HAUTE.

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
Nombre de jeux de test	2	2	1	1

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

– Enregistrer l'occupation d'une chambre par un client : **priorité haute**

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	F	F	F	F	F	F	F	F	T
Nombre de jeux de test	2	2	2	1	1	1	1	2	1

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
Nombre de jeux de test	2	1	1	1	1

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

3.2 Diagramme de classes

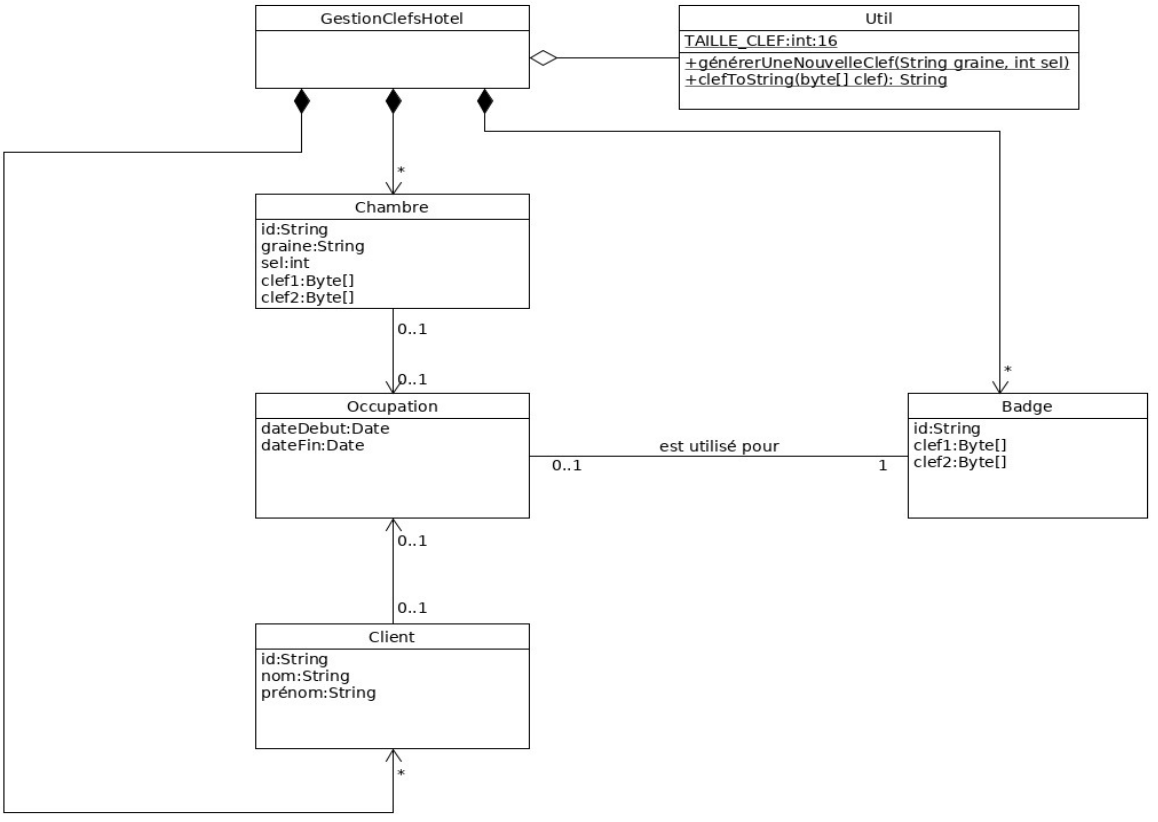

FIG. 1: Diagramme de classes

3.4 Diagrammes de séquence

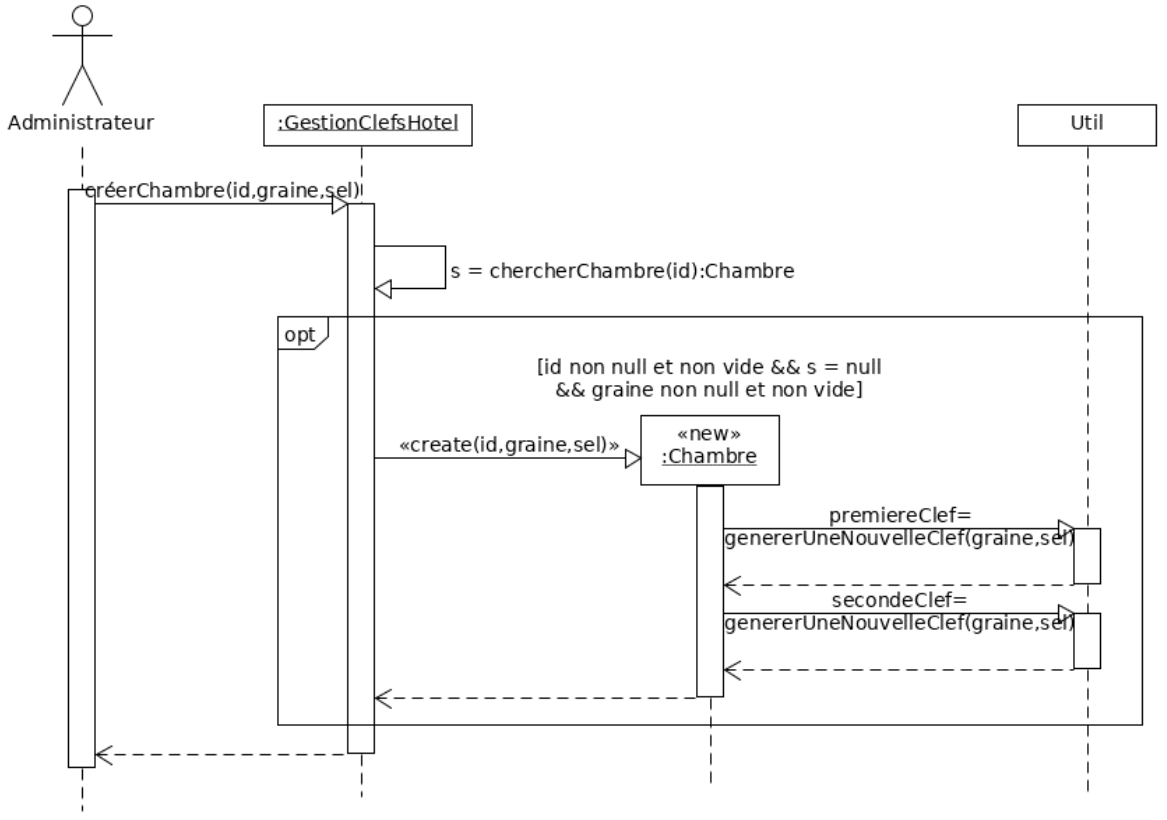

FIG. 1: Cas d'utilisation «créer une chambre»

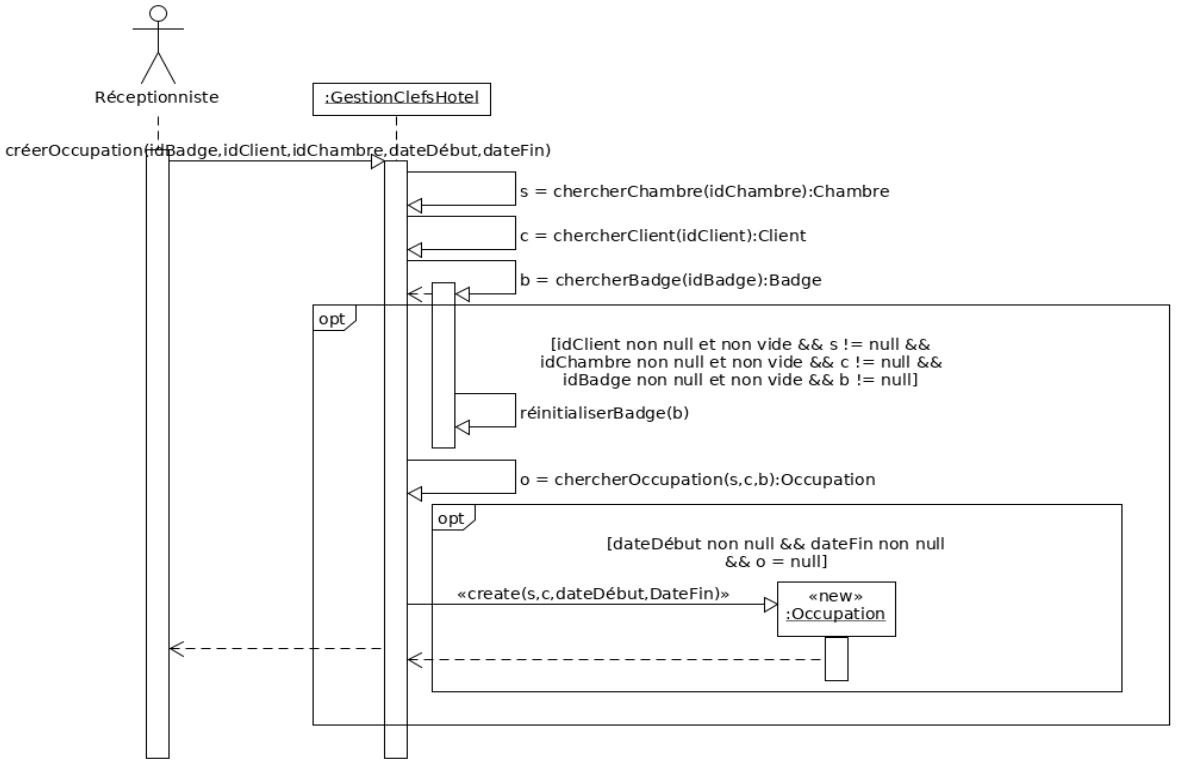

FIG. 2: Cas d'utilisation «créer une occupation»

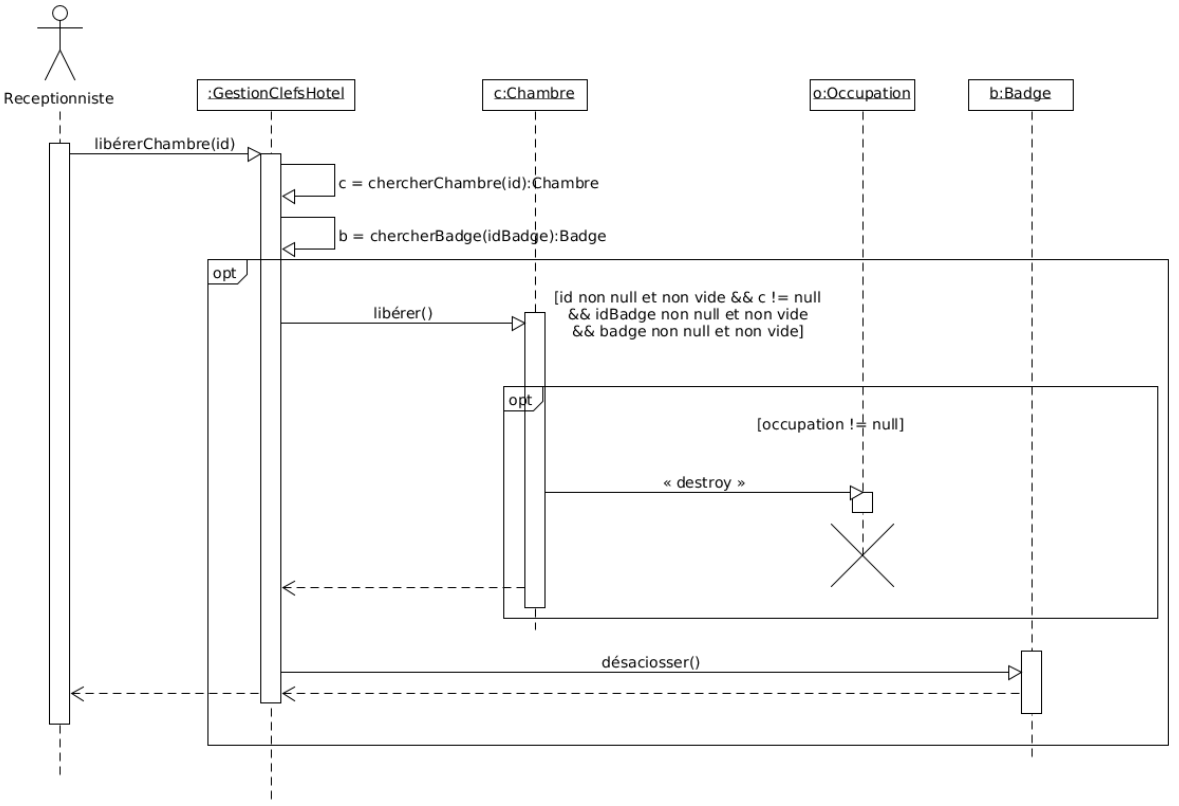

FIG. 3: Cas d'utilisation «libérer une chambre»

4 Fiche des classes

4.1 Classe GestionClefsHotel

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

4.2 Classe Badge

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

5 Diagrammes de machine à états et invariants

Les invariants suivants sont à compléter avec les diagrammes de machine à états et les invariants de vos classes les plus importantes.

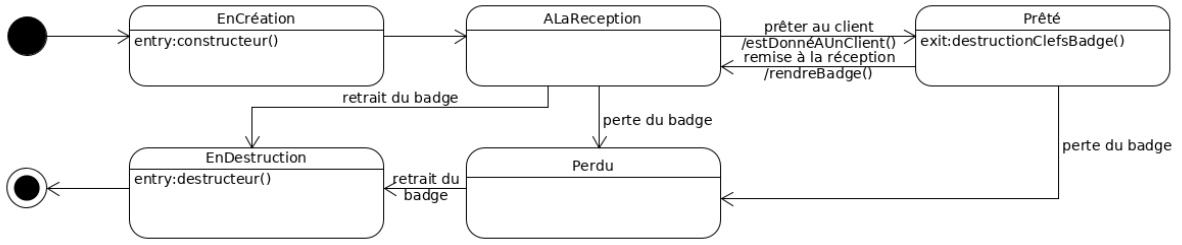

FIG. 1: Diagramme de machine à états «Badge d'accès»

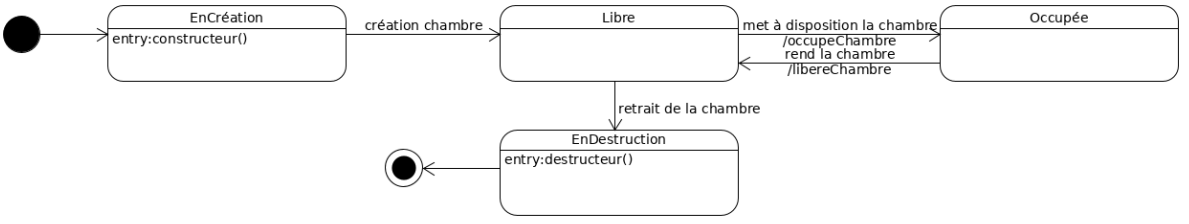

FIG. 2: Diagramme de machine à états «Chambre»

6 Préparation des tests unitaires

6.1 Classe Chambre

Invariant : Libre ∨ Occupée

6.2 Classe Badge

Invariant : (¬Prêté ∧ ¬Perdu) ∨ (Prêté ∧ ¬Perdu) ∨ (Prêté ∧ Perdu)

Numéro de test	1	2	3	4
Badge ≠ null	F	T	T	T
client ≠ null		F	T	T
chambre ≠ null			F	T
clef1’ = clef2				T
clef2’ ≠ clef2				T
invariant				T
Levée exception	OUI	OUI	OUI	NON
Objet crée	F	F	F	T
Nb jeux de test	1	1	1	1

Tableau 1 : Méthode « estDonnéAUnClient » de la classe Badge

Numéro de test	1	2	3	4	5
identifiant ≠ null ∧ ¬vide	F	T	T	T	T
occupation ≠ null		F	T	T	T
clef1 ≠ null			F	T	T
clef2 ≠ null				F	T
identifiant’ = identifiant					T
occupation.badge’ = badge					T
badge.occupation’ = occupation					T
clef1’ = clef1					T
clef2’ = clef2					T
invariant					T
Levée exception	OUI	OUI	OUI	OUI	NON
Objet crée	F	F	F	F	T
Nb jeux de test	2	1	1	1	1

Tableau 2 : Méthode « constructeur» de la classe Badge

Annexe : Algorithmes Badge

1 ALGO1C1 : constructeur

badge(String identifiant, Occupation o, Byte[] c1, Byte[] c2)

si identifiant == null ou identifiant = "" alors lever une exception

si o == null alors lever une exception

si c1 == null alors lever une exception

si c2 == null alors lever une exception

o.badge = identifiant

badge.occupation = occupation

id := identifiant

clef1 := c1

clef2 := c2

assert invariant()

2 ALGO2C1 : estDonnéAUnClient

estDonnéAUnClient(Badge b, Client c, Chambre ch, DateTime dateDebut, DateTime dateFin)

Si b == null alors lever une exception

si c == null alors lever une exception

si ch == null alors lever une exception

b.clef1 = b.clef2

b.clef2 = générerUneNouvelleClef(ch.graine, ch.sel + 1)

créerOccupation(b.id, ch.id, c.id, dateDebut, dateFin)

assert invariant()